Package 'riverdist'

January 4, 2022

Type Package

Title River Network Distance Computation and Applications

Version 0.15.5

Date 2021-12-31

Author Matt Tyers [aut, cre]

Maintainer Matt Tyers <matttyersstat@gmail.com>

Description Reads river network shape files and computes network distances.
Also included are a variety of computation and graphical tools designed
for fisheries telemetry research, such as minimum home range, kernel density
estimation, and clustering analysis using empirical k-functions with
a bootstrap envelope. Tools are also provided for editing the river
networks, meaning there is no reliance on external software.

License GPL-2

Depends R (>= 2.14.0)

Imports rgdal (>= 0.9-1), sp (>= 1.0-15), methods

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

LazyData TRUE

URL https://cran.r-project.org/package=riverdist

BugReports https://github.com/mbtyers/riverdist/issues

RoxygenNote 7.1.2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-01-04 17:20:05 UTC

R topics documented:

riverdist-package 3
abstreams 5

abstreams0 6
addcumuldist 6
addverts 7
buildlookup 8
buildsegroutes 9
calculateconnections 10
checkbraided 11
checkbraidedTF 12
cleanup 13
cleanup_verts 14
connectsegs 15
detectroute 16
dissolve 18
fakefish 19
fakefish_density 19
Gulk 20
highlightseg 20
homerange 21
homerange-class 22
homerangeoverlap 23
isflowconnected 24
Kenai1 25
Kenai2 25
Kenai3 26
kfunc 26
KilleyW 28
Koyukuk0 29
Koyukuk1 29
Koyukuk2 30
line2network 30
line98 32
makeriverdensity 32
mapbyname 34
matbysurveylist 34
mouthdist 36
mouthdistbysurvey 37
pdist 39
pdisttot 39
plot.homerange 40
plot.riverdensity 41
plot.rivernetwork 44
plothomerangeoverlap 45
plotmatbysurveylist 46
plotriverdensitypoints 47
plotseq 48
pointshp2segvert 49
removeduplicates 50
removemicrosegs 51

removeunconnected 52
riverdensity 52
riverdirection 53
riverdirectionmat 55
riverdirectionmatbysurvey 56
riverdirectionseq 58
riverdirectiontofrom 59
riverdistance 61
riverdistancelist 63
riverdistancemat 64
riverdistancematbysurvey 66
riverdistanceseq 67
riverdistancetofrom 69
rivernetwork 71
riverpoints 72
routelist 73
sequenceverts 74
setmouth 75
showends 76
smallset 77
splitsegmentat 77
splitsegments 78
topologydots 80
trimriver 80
trimtopoints 81
upstream 83
upstreammat 84
upstreammatbysurvey 86
upstreamseq 87
upstreamtofrom 89
whoconnected 91
xy2segvert 92
zoomtoseg 93

Index **95**

`riverdist-package` *River Network Distance Computation and Applications*

Description

Reads river network shape files and computes network distances. Also included are a variety of computation and graphical tools designed for fisheries telemetry research, such as minimum home range, kernel density estimation, and clustering analysis using empirical k-functions with a bootstrap envelope. Tools are also provided for editing the river networks, meaning there is no reliance on external software.

Details

Package:	riverdist
Type:	Package
Version:	0.15.5
Date:	2021-12-31
License:	GPL-2

The riverdist package provides tools for distance calculation along a river network. The river network is imported from a projected shapefile. Spatial point data may be imported from a shapefile as well, or directly from coordinates.

Some basic formatting of the river shapefile may be necessary. If available, formatting in a geographic information system (GIS) prior to importing into R is recommended (projecting, spatial trimming to the study area, and possibly dissolving river segments), but the riverdist package and its dependencies also include tools for accomplishing the necessary formatting within R.

Author(s)

Matt Tyers

Maintainer: Matt Tyers <matttyersstat@gmail.com>

`abstreams` *Dataset: A-B Streams*

Description

A complex river network object, a subset of the streams in the Absaroka-Beartooth Wilderness.

Usage

```
data(abstreams)
```

Format

A river network object, see rivernetwork

abstreams0 *Dataset: A-B Streams 0*

Description

An unusably messy river network object, included for the purpose of testing river network editing functions.

Usage

```
data(abstreams0)
```

Format

A river network object, see rivernetwork

addcumuldist *Add Cumulative Distance to a River Network*

Description

Adds a vector of cumulative distances to a river network. Called internally.

Usage

```
addcumuldist(rivers)
```

Arguments

`rivers`	The river network object to use.

Value

Returns an object of class `"rivernetwork"` containing all spatial and topological information. See rivernetwork-class.

Author(s)

Matt Tyers

Examples

```
Gulk1 <- addcumuldist(rivers=Gulk)
```

addverts *Add Vertices To Maintain a Minimum Distance Between Vertices*

Description

In certain cases, such as when there is a lake within a river system, there may be long, straight lines in a river network with vertices only at either end. In these cases, any point data along these stretches would be snapped to the vertices at either end. This function automatically adds equally-spaced vertices along the straight line, according to a specified minimum allowable distance between vertices.

Usage

```
addverts(rivers, mindist = 500)
```

Arguments

`rivers`	The river network object to use.
`mindist`	The minimum distance to use between vertices. Defaults to 500.

Value

A new river network object with the specified segments connected (see rivernetwork)

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Kenai3)
Kenai3split <- addverts(Kenai3,mindist=200)

zoomtoseg(seg=c(47,74,78), rivers=Kenai3)
points(Kenai3$lines[[74]])        # vertices before adding

zoomtoseg(seg=c(47,74,78), rivers=Kenai3split)
points(Kenai3split$lines[[74]])   # vertices after adding
```

buildlookup — *Build Lookup Tables for Fast Distance Computation*

Description

Adds lookup tables for distance computation, dramatically reducing computation time. It may take some time to calculate, particularly in a braided network.

Usage

```
buildlookup(rivers)
```

Arguments

`rivers`	The river network object to use

Value

A rivernetwork object, with a new list element, `$distlookup`, a list of three matrices. Element `[i,j]` of each matrix corresponds to the route between segment `i` and `j`. The `distlookup$middist` matrix gives the total distance of the "middle" of each route (between the starting and ending segments"), and the `distlookup$starttop` and `distlookup$endtop` matrices have value `TRUE`, `FALSE`, or `NA` if the segments at the beginning or end of the route are connected to the rest of the route at the top of the coordinate matrix, bottom of the coordinate matrix, or if the route is contained to just one segment, respectively. (See rivernetwork.)

Note

This will add three n by n matrices to the river network object, which will be very large if the river network has many segments.

This function is called within cleanup, which is recommended in most cases. It is also called within buildsegroutes, and will add lookup tables by default if there are fewer than 400 segments in the river network.

This function can still be called in the presence of a braided network, but all resulting distances used in subsequent analyses will be the shortest route.

If segment routes (`$segroutes`) are not present, this function may take a very long time to run.

Author(s)

Matt Tyers

Examples

```
data(abstreams)

abstreams1 <- buildlookup(abstreams)
```

`buildsegroutes` *Build Segment Routes*

Description

Adds the travel routes from the mouth (lowest point) of a river network to each segment, and (optionally) distance lookup tables. This greatly reduces the time needed to detect routes, making distance calculation much more efficient, particularly in the case of multiple distance calculations.

Usage

```
buildsegroutes(rivers, lookup = NULL, verbose = FALSE)
```

Arguments

`rivers`	The river network object to use
`lookup`	Whether to build lookup tables as well. This may take some time, but will result in even faster distance computation in analyses (see buildlookup). Because of the object size returned, this may not be advisable in a large river network (more than a few hundred segments). Accepts `TRUE` or `FALSE`, and defaults to `NULL`. If the default value is accepted, lookup tables will be built if the river network has 400 segments or fewer.
`verbose`	Whether or not to print the segment number the function is currently building a route for (used for error checking). Defaults to FALSE.

Value

A rivernetwork object, with a new list element, `$segroutes`, which gives the route from the mouth to each rivernetwork segment. Optionally, it may add `$distlookup`, distance lookup tables for even faster distance computation. (See rivernetwork.)

Note

In the event of braiding (multiple channels), it is likely that there will be differences in the routes detected. If this is the case, building routes will likely result in a shorter and more efficient route. Regardless, extreme caution is always advised in the event of braiding.

The mouth segment and vertex must be specified (see setmouth).

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

Examples

```
data(abstreams)
plot(x=abstreams)
abstreams1 <- abstreams
abstreams1$segroutes <- NULL #taking out the $segroutes component

# before
tstart <- Sys.time()
detectroute(start=120, end=111, rivers=abstreams1)
Sys.time() - tstart

# after
tstart <- Sys.time()
detectroute(start=120, end=111, rivers=abstreams)
Sys.time() - tstart
```

---

calculateconnections *Calculate the Connectivity Matrix for a River Network*

Description

Calculates the connectivity matrix for a river network, during import and editing. Called internally.

Usage

```
calculateconnections(lines, tolerance)
```

Arguments

`lines`	A list of coordinate matrices, each corresponding to a line segment.
`tolerance`	The spatial tolerance for establishing connectivity.

Value

A matrix with topological information. See the `$connections` element of the rivernetwork-class.

Author(s)

Matt Tyers

Examples

```
Gulk_connections <- calculateconnections(lines=Gulk$lines, tolerance=Gulk$tolerance)
```

`checkbraided` *Check for Braiding in a River Network*

Description

Detects braiding (multiple flow channels between two locations) within a river network object. Braiding can either be checked for in the route between two segments, or in the river network as a whole.

Usage

```
checkbraided(rivers, startseg = NULL, endseg = NULL, progress = TRUE)
```

Arguments

`rivers`	The river network object to check.
`startseg`	Starting segment of a route to investigate. If this and `endseg` are `NULL`, the full river network will be checked.
`endseg`	Starting segment of a route to investigate. If this and `startseg` are `NULL`, the full river network will be checked.
`progress`	Whether to show the progress bar. Defaults to `TRUE`.

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

Examples

```
data(Gulk)
plot(x=Gulk)
checkbraided(rivers=Gulk)

data(KilleyW)
plot(x=KilleyW)
checkbraided(rivers=KilleyW)

Kenai3.subset <- trimriver(trimto=c(22,2,70,30,15,98,96,89,52,3), rivers=Kenai3)
plot(x=Kenai3.subset)

checkbraided(startseg=1, endseg=7, rivers=Kenai3.subset)
checkbraided(startseg=1, endseg=5, rivers=Kenai3.subset)
```

checkbraidedTF — *Check for Braiding in a River Network*

Description

Detects braiding (multiple flow channels between two locations) within a river network object, and returns a logical value for specifying braiding within a river network object.

Usage

```
checkbraidedTF(rivers, toreturn = "rivers", progress = TRUE)
```

Arguments

`rivers`	The river network object to check.
`toreturn`	Specifying `toreturn="rivers"` (the default) will return a river network object with a value of `TRUE` or `FALSE` assigned to the `$braided` element of the river network object. Specifying `toreturn="logical"` will just return `TRUE` if braiding is detected or `FALSE` if no braiding is detected. Specifying `toreturn="routes"` will return the first two differing routes detected, which may be useful in identifying where the problem lies.
`progress`	Whether to show the progress bar. Defaults to `TRUE`.

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

Examples

```
data(Gulk,KilleyW)
Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)
plot(x=Gulk)
checkbraidedTF(rivers=Gulk, toreturn="logical")

KilleyW <- setmouth(seg=1, vert=288, rivers=KilleyW)
plot(x=KilleyW)
checkbraidedTF(rivers=KilleyW, toreturn="logical")
checkbraidedTF(rivers=KilleyW, toreturn="routes")

KilleyW.1 <- checkbraidedTF(rivers=KilleyW, toreturn="rivers")
str(KilleyW.1)
```

`cleanup` *Interactive Cleanup of a River Network*

Description

This is the recommended function to use for cleanup of a river network. It calls all available river network editing functions in appropriate sequence, detecting which are needed or recommended, and prompts user input wherever necessary.

Currently, it automatically calls removeduplicates, prompts the user whether to run dissolve, automatically runs removemicrosegs and splitsegments if needed, provides user prompts for addverts and setmouth, detects if segments are unconnected and provides user prompts for removeunconnected or connectsegs, automatically runs checkbraidedTF, and prompts the user whether to run buildsegroutes if no braiding is detected.

Usage

```
cleanup(rivers)
```

Arguments

`rivers`	The river network object to use

Value

A new river network object, see rivernetwork

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(abstreams0,Koyukuk0,Kenai1)

# abstreams_fixed <- cleanup(abstreams0)
# Koyukuk <- cleanup(Koyukuk0)
# Kenai <- cleanup(Kenai1)
```

`cleanup_verts` *Interactive Cleanup of the Vertices of Individual Segments*

Description

A trial version of a function for deep-cleaning a river network.

Sometimes a shapefile contains errors that are not obvious at an initial check, typically vertices that should not be there.

This function steps through each segment in sequence, and allows the user to interactively remove vertices.

Usage

```
cleanup_verts(rivers, startwith = 1)
```

Arguments

`rivers`	The river network object to use
`startwith`	The segment (number) to start with, defaulting to `1`.

Value

A new river network object, see rivernetwork

Note

Stepping through a large and messy river network can be time-consuming. To resume a cleanup session, use the `startwith=` argument and the last returned river network. For example, if `rivers1 <-cleanup_verts(rivers)` were initially called and the user selected "save & close" at segment 100, cleanup can be resumed by calling `rivers2 <-cleanup_verts(rivers1,startwith=100)`.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(abstreams0,Koyukuk0,Kenai1)

# abstreams_fixed1 <- cleanup_verts(abstreams0)
# Koyukuk <- cleanup(Koyukuk0)
# Kenai <- cleanup(Kenai1)
```

connectsegs *Connect Segments*

Description

Provides a method to manually connect unconnected segments within a river network. The nearest endpoint (or vertex) of the second segment is added as a new vertex to the first, and the network topology is then updated.

Usage

```
connectsegs(
  connect,
  connectto,
  nearestvert = TRUE,
  rivers,
  calcconnections = TRUE
)
```

Arguments

`connect`	The segment(s) to connect to the network. Typically, this is the segment that is disconnected from the rest of the river network. A vector of segments may be used.
`connectto`	The segment(s) to connect it (them) to. Typically, this is a segment that is connected to the rest of the river network. A vector of segments may be used, corresponding to that used in `connect=`.
`nearestvert`	Whether to connect at the nearest vertex and split the segment (`FALSE`), or connect at the nearest endpoint (`TRUE`). Defaults to `TRUE`. A vector may be used, corresponding to those used in `connect=` and `connectto=`.
`rivers`	The river network object to use.
`calcconnections`	Whether to recalculate all connections. Defaults to `TRUE`. Setting to `FALSE` is not recommended unless many connections are to be made, in which case connections can be calculated afterward.

Value

A new river network object with the specified segments connected (see rivernetwork)

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Koyukuk0)
plot(Koyukuk0, ylim=c(1930500,1931500), xlim=c(194900,195100))
topologydots(Koyukuk0, add=TRUE)

Koyukuk0.1 <- connectsegs(connect=21, connectto=20, rivers=Koyukuk0)
plot(Koyukuk0.1,ylim=c(1930500,1931500), xlim=c(194900,195100))
topologydots(Koyukuk0.1, add=TRUE)

# or the vector version
zoomtoseg(seg=21:23, rivers=Koyukuk0)
Koyukuk0.2 <- connectsegs(connect=c(20,21,22), connectto=c(21,22,23),
    nearestvert=c(FALSE,FALSE,TRUE), rivers=Koyukuk0)
zoomtoseg(seg=21:23, rivers=Koyukuk0.2)
topologydots(Koyukuk0.2, add=TRUE)
```

`detectroute` *Detect Route*

Description

Called internally within riverdistance. Detects the sequential route from one river network segment to another.

Usage

```
detectroute(
  start,
  end,
  rivers,
  verbose = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`start`	Segment number of the start of the route
`end`	Segment number of the end of the route
`rivers`	The river network object to use
`verbose`	Whether or not to print all routes being considered (used for error checking). Defaults to FALSE.

`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, `detectroute()` will return `NA`. Defaults to `TRUE`.
`algorithm`	Which route detection algorithm to use. If set to `NULL` (the default), the function will automatically make a selection. Choices are: • Setting `algorithm="sequential"` will be quite slow, and may give inaccurate results in the event of braiding. This algorithm returns the first complete route detected, which may not be the shortest. This algorithm is not recommended in almost all cases, but is retained as an option for certain checks. It will not be used unless specified. • Setting `algorithm="Dijkstra"` will be much faster, and will return the shortest route in the event of braiding. If braiding is present or unknown, this will be the algorithm automatically chosen. • Setting `algorithm="segroutes"` will be the fastest of all, but will only return results in a non-braided network. This will be the algorithm automatically selected if segment routes are present - see buildsegroutes.

Value

A vector of segment numbers corresponding to the ordered route.

Author(s)

Matt Tyers

Examples

```
data(Gulk)
plot(x=Gulk, cex=1)

detectroute(start=6, end=14, rivers=Gulk)

tstart <- Sys.time()
detectroute(start=120, end=111, rivers=abstreams, algorithm="sequential")
tend <- Sys.time()
tend - tstart

data(abstreams)
tstart <- Sys.time()
detectroute(start=120, end=111, rivers=abstreams, algorithm="Dijkstra")
tend <- Sys.time()
tend - tstart

tstart <- Sys.time()
detectroute(start=120, end=111, rivers=abstreams, algorithm="segroutes")
tend <- Sys.time()
tend - tstart
```

`dissolve`	*Dissolve*

Description

Acts like a spatial dissolve within a GIS environment. Simplifies a river network object by combining "runs" of segments with no other connections.

Usage

```
dissolve(rivers)
```

Arguments

`rivers`	The river network object to use

Value

A new river network object with segments combined

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Kenai2)
plot(x=Kenai2)

Kenai2dissolve <- dissolve(rivers=Kenai2)
plot(x=Kenai2dissolve)
```

`fakefish` *Dataset: Fakefish*

Description

A set of observations of Fakefish on the Gulkana River and its tributaries.

Usage

```
data(fakefish)
```

Format

A data frame

Details

- `x`. X-coordinate of observation (Alaska Albers Equal Area). Note that the locations do not align with the river network object.
- `y`. Y-coordinate of observation
- `seg`. River segment (with x- and y-coordinates snapped to river network object)
- `vert`. River vertex
- `fish.id`. Numeric identifier for each fish (individual fish were observed more than once)
- `flight`. Numeric identifier for each telemetry flight
- `flight.date`. Date of each telemetry flight

See Also

Gulk

`fakefish_density` *Dataset: Fakefish Density*

Description

An object created by riverdensity, describing the density of Fakefish points in the Gulkana River during ten surveys.

Usage

```
data(fakefish_density)
```

Format

A river density object, see riverdensity, plotriverdensity, riverdensity-class

Details

Intended for plotting using plotriverdensity.

`Gulk` *Dataset: Gulkana River*

Description

A stretch of Gulkana River and tributaries.

Usage

```
data(Gulk)
```

Format

A river network object, see rivernetwork

`highlightseg` *Highlight Segments*

Description

Plots a river network object and displays specified segments in bold, for easy identification.

Usage

```
highlightseg(seg, rivers, cex = 0.8, lwd = 3, add = FALSE, color = FALSE, ...)
```

Arguments

`seg`	A vector of segments to highlight
`rivers`	The river network object to use
`cex`	The character expansion factor to use for segment labels
`lwd`	The line width to use for highlighted segments
`add`	Whether to add the highlighted segments to an existing plot (`TRUE`) or call a new plot (`FALSE`). Defaults to `FALSE`.
`color`	Whether to display segment labels as the same color as the segments. Defaults to `FALSE`.
`...`	Additional plotting arguments (see par)

Author(s)

Matt Tyers

Examples

```
data(Kenai3)
plot(Kenai3)
highlightseg(seg=c(10,30,68),rivers=Kenai3)
```

---

homerange *Home Range*

Description

Returns the minimum observed home range for multiple observations of each individual fish.

Usage

```
homerange(
  unique = NULL,
  survey = NULL,
  seg,
  vert,
  rivers,
  map = FALSE,
  algorithm = NULL,
  main = NULL,
  ...
)
```

Arguments

`unique`	A vector of unique identifiers for each fish. If the default (`NULL`) is used, the function will assume all observations come from a single individual.
`survey`	A vector of survey identifiers for each fish. This argument is not needed for home range calculation, but can affect plotting (see plot.homerange).
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`map`	Deprecated, use plot.homerange for plotting instead. Originally, whether to produce sanity-check maps of observed locations and calculated home range for each fish.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.
`main`	Deprecated, use plot.homerange for plotting instead. Originally, plot title, if `map` is set to `TRUE`. If unspecified, the unique ID will be used for the title.
`...`	Deprecated, use plot.homerange for plotting instead. Originally, additional plotting arguments, if `map` is set to `TRUE`.

Value

An object of the homerange-class. The `$ranges` element is a data frame with two columns: `$ID` is a list of unique fish (as specified by `unique=`), and `$range` is calculated minimum home range, in the units of the coordinate system (this will likely be meters). The other elements are used for plotting, see homerange-class for more details.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

plot.homerange, homerangeoverlap, plothomerangeoverlap

Examples

```
data(Gulk, fakefish)
ranges <- with(fakefish, homerange(unique=fish.id, survey=flight, seg=seg, vert=vert, rivers=Gulk))
ranges

# 19 plots will be produced, recommend calling par(mfrow=c(4,5))
plot(ranges)
plot(ranges,cumulative=TRUE,label=TRUE)

homerangeoverlap(ranges)

plothomerangeoverlap(ranges)
with(fakefish, riverpoints(seg=seg, vert=vert, rivers=Gulk))
```

`homerange-class` *The "homerange" Class*

Description

A class that holds information computed from the homerange function. Contains all information for plotting in plot.homerange.

Elements

`ranges`**:** Object of class `"data.frame"`. Contains a column of the identifiers for each individual, and a column of the associated home ranges.

`subseg_n`**:** List of the number of times each subsegment was traveled. The first level of the list corresponds to individual, the second level to river segment.

`subseg_length`**:** List of lengths of each subsegment.

`seg, vert, unique, rivers`**:** All inputs from the original homerange call.

Author(s)

Matt Tyers

`homerangeoverlap` *Home Range Overlap*

Description

Returns matrices describing the overlap of the minimum observed home range for multiple observations of each individual fish.

Usage

```
homerangeoverlap(x)
```

Arguments

`x`	An object returned from homerange.

Value

A list of three matrices, with `$either` giving the distances represented by the union of home ranges of each pair of individuals, and `$both` giving the distances represented by the intersection of home ranges of each pair of individuals. Element `$prop_both` gives the proportion of overlap, defined as intersection/union.

Author(s)

Matt Tyers

See Also

homerange, plot.homerange, plothomerangeoverlap

Examples

```
data(Gulk, fakefish)
ranges <- with(fakefish, homerange(unique=fish.id, survey=flight, seg=seg, vert=vert, rivers=Gulk))
ranges

# 19 plots will be produced, recommend calling par(mfrow=c(4,5))
plot(ranges)
plot(ranges,cumulative=TRUE,label=TRUE)

homerangeoverlap(ranges)
```

```
plothomerangeoverlap(ranges)
with(fakefish, riverpoints(seg=seg, vert=vert, rivers=Gulk))
```

isflowconnected *Check Flow-Connectedness*

Description

Checks to see if two segments are flow-connected. Called internally within riverdirection and upstream.

Usage

```
isflowconnected(seg1, seg2, rivers, stopiferror = TRUE, algorithm = NULL)
```

Arguments

`seg1`	First input segment
`seg2`	Second input segment
`rivers`	The river network object to use
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

Logical `TRUE` if the two segments are flow-connected, `FALSE` if they are not

Note

The river mouth must be specified (see setmouth).

Author(s)

Matt Tyers

Examples

```
data(Gulk)
plot(Gulk)

Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)

isflowconnected(seg1=13, seg2=14, rivers=Gulk)
isflowconnected(seg1=13, seg2=1, rivers=Gulk)
```

`Kenai1` *Dataset: Kenai River 1*

Description

A first pass at a messy river network object.

Usage

```
data(Kenai1)
```

Format

A river network object, see rivernetwork

See Also

Kenai2, Kenai3

`Kenai2` *Dataset: Kenai River 2*

Description

A second pass at a messy river network object. In this iteration of cleanup, several non-connected segments have been removed.

Usage

```
data(Kenai2)
```

Format

A river network object, see rivernetwork

See Also

Kenai1, Kenai3

Kenai3 *Dataset: Kenai River 3*

Description

A third pass at a messy river network object. In this iteration of cleanup, several non-connected segments have been removed, and several series of segments have been dissolved into single segments.

Usage

```
data(Kenai3)
```

Format

A river network object, see rivernetwork

See Also

Kenai1, Kenai2

kfunc *Plotting K-functions for a Set of Surveys*

Description

Plots K-functions for locations in each of a set of surveys. In this implementation, this can be interpreted as the proportion of additional fish within a given distance. This will increase as a function of distance, and may provide evidence of clustering or dispersion features, particularly if the envelope is used.

Usage

```
kfunc(
  seg,
  vert,
  survey = NULL,
  rivers,
  lwd = 2,
  envelope = TRUE,
  envreps = 1000,
  envcol = "grey80",
  envborder = NA,
  maxdist = NULL,
  xlab = "Distance",
  ylab = "% within",
  showN = TRUE,
  whichplots = NULL,
  returnoutput = FALSE,
  ...
)
```

Arguments

`seg`	A vector of river locations (segment)
`vert`	A vector of river locations (vertex)
`survey`	A vector of survey IDs corresponding to the values of `seg` and `vert`. Defaults to `NULL`. If this argument is used, K-functions will be calculated for each unique survey, and separate plots will be produced.
`rivers`	The river network object to use
`lwd`	Line width used for plotting. Defaults to 2.
`envelope`	Whether to construct and display a 95 percent confidence envelope (see note.) Defaults to `TRUE` if `survey` is specified, and is automatically `FALSE` otherwise.
`envreps`	Number of bootstrap replicates to use for envelope calculation. Defaults to 1000.
`envcol`	Color to use for envelope plotting. Defaults to `"grey80"`.
`envborder`	Border color to use for envelope plotting. Defaults to `NA`, which will result in no border being plotted.
`maxdist`	Maximum distance (x-axis value) for plotting. The default value (`NULL`) will result in an appropriate value being chosen.
`xlab`	X-coordinate label for plotting
`ylab`	Y-coordinate label for plotting
`showN`	Whether to show the sample size for each survey in each plot title. Defaults to `TRUE`.
`whichplots`	A vector of plots to produce, if multiple plots are produced. For example, specifying `whichplot=c(2,3,4)` will result in only the second, third, and fourth plots of the sequence being produced. Accepting the default (`NULL`) will result in all plots being produced.
`returnoutput`	Whether to return output instead of producing a plot. Defaults to `FALSE`.
`...`	Additional plotting parameters.

Note

K-function envelopes for each survey are constructed by bootstrapping all within-survey distances, that is, the distances between all individuals within each survey, for all surveys. This results in a confidence envelope under the assumption that spacing is independent of survey; therefore a survey K-function outside the envelope provides evidence of clustering or dispersal in that survey that is outside the typical range. An envelope is not available if only one survey is plotted.

A K-function above the envelope for a given distance range provides evidence of a greater number of individuals than expected at that distance range (clustering); A K-function below the envelope for a given distance range provides evidence of a smaller number of individuals than expected at that distance range (dispersal).

This function is distance-computation intensive, and will be extremely slow-running if a river network is used that does not have segment routes and/or distance lookup tables for fast distance computation. See buildsegroutes and/or buildlookup for more information.

Author(s)

Matt Tyers

Examples

```
data(Gulk, fakefish)

# # 10 plots will be created - recommend calling
# # par(mfrow=c(3,4))

kfunc(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, survey=fakefish$flight,
envreps=100, maxdist=200000)

# # This shows relatively high amounts of clustering for surveys 1 and 8,
# # and relatively high amounts of dispersal in surveys 5 and 6.


# # plotting the survey locations that led to this calculation, for comparison

# # 10 plots will be created - recommend calling
# # par(mfrow=c(3,4))
for(i in 1:10) {
  plot(x=Gulk, segmentnum=FALSE, color=FALSE, main=i)
  riverpoints(seg=fakefish$seg[fakefish$flight==i],
  vert=fakefish$vert[fakefish$flight==i], rivers=Gulk, col=2, pch=15)
}
```

`KilleyW` *Dataset: Killey River, West Channel*

Description

A messy and braided section of the Kenai River network - actually a subset of Kenai3.

Usage

```
data(KilleyW)
```

Format

A river network object, see rivernetwork

`Koyukuk0` *Dataset: Koyukuk River 0*

Description

An unusably messy river network object, included for the purpose of testing river network editing functions.

Usage

```
data(Koyukuk0)
```

Format

A river network object, see rivernetwork

See Also

Koyukuk1, Koyukuk2

`Koyukuk1` *Dataset: Koyukuk River 1*

Description

A first pass at a messy river network object. The way it was dissolved in ArcGIS makes the end-points appear disconnected to line2network and the topologies do not work.

Usage

```
data(Koyukuk1)
```

Format

A river network object, see rivernetwork

See Also

Koyukuk2

Koyukuk2 *Dataset: Koyukuk River 2*

Description

A second pass at a messy river network object, with topologies fixed from Koyukuk1.

Usage

```
data(Koyukuk2)
```

Format

A river network object, see rivernetwork

See Also

Koyukuk1

line2network *Create a River Network Object from a Shapefile*

Description

Uses readOGR in package 'rgdal' to read a river shapefile, and establishes connectivity of segment endpoints based on spatial proximity.

Usage

```
line2network(
  sp = NA,
  path = ".",
  layer = NA,
  tolerance = 100,
  reproject = NULL,
  supplyprojection = NULL
)
```

Arguments

`sp`	SpatialLinesDataFrame object. optional.
`path`	File path, default is the current working directory.
`layer`	Name of the shapefile, without the .shp extension.

`tolerance`	Snapping tolerance of segment endpoints to determine connectivity. Default is 100, therefore care should be exercised when working with larger units of distance, such as km.
`reproject`	A valid Proj.4 projection string, if the shapefile is to be re-projected. Re-projection is done using spTransform in package 'sp'.
`supplyprojection`	A valid Proj.4 projection string, if the input shapefile does not have the projection information attached.

Value

Returns an object of class `"rivernetwork"` containing all spatial and topological information. See rivernetwork-class.

Note

Since distance can only be calculated using projected coordinates, `line2network()` will generate an error if a non-projected input shapefile is detected. To resolve this, the shapefile can be re-projected in a GIS environment, or using `reproject=`, shown in the second example below.

Author(s)

Matt Tyers, Joseph Stachelek

Examples

```
filepath <- system.file("extdata", package="riverdist")

Gulk_UTM5 <- line2network(path=filepath, layer="Gulk_UTM5")
plot(Gulk_UTM5)

## Reading directly from a SpatialLinesDataFrame object

sp <- rgdal::readOGR(dsn = filepath, layer = "Gulk_UTM5", verbose = FALSE)
Gulk_UTM5 <- line2network(sp)
plot(Gulk_UTM5)

## Re-projecting in Alaska Albers Equal Area projection:

AKalbers <- "+proj=aea +lat_1=55 +lat_2=65 +lat_0=50 +lon_0=-154
    +x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80"

Gulk_AKalbers <- line2network(path=filepath, layer="Gulk_UTM5", reproject=AKalbers)
plot(Gulk_AKalbers)
```

line98 *Dataset: Line 98 of Kenai River 1 (Long-Lat)*

Description

A matrix of coordinates in longitude-latitude, used to illustrate coordinate transformation. Coordinates come from arbitrary line number 98 in the Kenai River 1 shapefile, rendered in long-lat.

Usage

```
data(line98)
```

Format

A matrix of values

makeriverdensity *Calculate Kernel Density Using River Distance*

Description

Uses spatial point data (segment and vertex) to calculate a kernel density object to use in the output class plotting method,plot.riverdensity. Scaled kernel density is calculated at approximately regularly-spaced locations, with spacing specified by the user.

If an argument is used in the `survey` field, kernel densities will be calculated for each unique value of `survey`, resulting in a separate plot for each.

The purpose of this function is to generate a kernel density object to plot using plot(), see plot.riverdensity.

Usage

```
makeriverdensity(
  seg,
  vert,
  rivers,
  survey = NULL,
  kernel = "gaussian",
  bw = NULL,
  resolution = NULL
)
```

Arguments

`seg`	A vector of river locations (segment)
`vert`	A vector of river locations (vertex)
`rivers`	The river network object to use
`survey`	A vector of survey IDs corresponding to the values of `seg` and `vert`. If this argument is used, kernel densities will be calculated for each unique survey, and separate plots will be produced.
`kernel`	The type of density kernel to use. Allowed types are `"gaussian"` (normal) and `"rect"` (rectangular, giving simple density). Defaults to `"gaussian"`.
`bw`	The kernel bandwidth to use. If `kernel` is set to `"gaussian"`, this provides the standard deviation of the gaussian (normal) kernel to use. If `kernel` is set to `"rect"`, this provides the half-width of the rectangular kernel, or the distance to use in simple density. Accepting the default (`NULL`) will result in the function determining a value to use, based on the total length of the river network and the value of the `resolution` argument.
`resolution`	The approximate spacing of the river locations used for kernel density calculation. Accepting the default (`NULL`) will result in the function determining a value to use, based on the total length of the river network.

Value

A river density object, see riverdensity-class.

Note

It is likely that calculation will be very slow. Use of this function with a river network for which segment routes has not yet been calculated is not recommended.

This function is distance-computation intensive, and may be slow-running if a river network is used that does not have segment routes and/or distance lookup tables for fast distance computation. See buildsegroutes and/or buildlookup for more information.

Author(s)

Matt Tyers

See Also

plot.riverdensity, plotriverdensitypoints

Examples

```
data(Gulk, fakefish)

Gulk_dens <- makeriverdensity(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk,
  survey=fakefish$flight.date)

# # 10 plots will be created, recommend calling par(mfrow=c(2,5))
plot(x=Gulk_dens)
```

mapbyname *Map Segments by Name*

Description

Provides a check that river network segments were appropriately named.

Usage

```
mapbyname(rivers, scale = TRUE, cex = 0.6, ...)
```

Arguments

`rivers`	The river network object to use. Function checks segment names contained in the river network object.
`scale`	Whether or not to give x- and y-axes the same scale
`cex`	Global character expansion factor for plotting
`...`	Additional plotting arguments (see par)

Author(s)

Matt Tyers

Examples

```
data(Gulk)
str(Gulk)

Gulk$names <- c("Gulkana River","Trib 1","West Fork","Gulkana River","Trib 1",
                "West Fork","Trib 2","West Fork","Twelvemile Creek","Gulkana River",
                "Middle Fork","Gulkana River","Middle Fork","Hungry Hollow")
str(Gulk)

mapbyname(rivers=Gulk)
```

matbysurveylist *Generate List of Distance Matrix Between Observations, for All Individuals*

Description

Returns a list of matrices, each giving the river distance, direction, or upstream travel distance between all observations of one unique fish. This function is principally intended for producing an object to plot in plotmatbysurveylist.

Usage

```
matbysurveylist(
  unique,
  survey,
  seg,
  vert,
  rivers,
  indiv = NULL,
  method = "upstream",
  flowconnected = FALSE,
  net = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`unique`	A vector of unique identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`indiv`	A vector of unique individuals to use. Accepting the default (`NULL`) will result in a matrix being returned for all unique individuals.
`method`	Which general method to use. Setting `method="distance"` will compute distance for each pair of observation, setting `method="direction"` will compute direction between each pair of observation, and setting `method="upstream"` will compute directional (upstream) distance between each pair of observation. Defaults to `"upstream"`.
`flowconnected`	Optional parameter to pass to the distance or direction calculation. Defaults to `FALSE`.
`net`	Optional parameter to pass to the distance or direction calculation. Defaults to `FALSE`.
`stopiferror`	Optional parameter to pass to the distance or direction calculation. Defaults to `TRUE`.
`algorithm`	Optional parameter to pass to the distance or direction calculation. Defaults to `NULL`.

Value

A list with each element corresponding to a unique fish. Each list element is the output from either riverdistancematbysurvey, riverdirectionmatbysurvey, or upstreammatbysurvey.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdistance, riverdirection, upstream, riverdistancematbysurvey, riverdirectionmatbysurvey, upstreammatbysurvey, plotmatbysurveylist

Examples

```
data(Gulk, smallset)
matbysurveylist <- matbysurveylist(unique=smallset$id, survey=smallset$flight, seg=smallset$seg,
   vert=smallset$vert, rivers=Gulk)
plotmatbysurveylist(matbysurveylist)
plotmatbysurveylist(matbysurveylist,type="confint")
plotmatbysurveylist(matbysurveylist,type="dotplot")

data(fakefish)
# matbysurveylist <- matbysurveylist(unique=fakefish$fish.id, survey=fakefish$flight,
#   seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)
# plotmatbysurveylist(matbysurveylist)
```

---

mouthdist *Distance From Mouth*

Description

Calculates distance from river locations (given as vectors of segment and vertex) and the specified mouth of the river network. The mouth must first be specified (see setmouth).

Usage

```
mouthdist(seg, vert, rivers, stopiferror = TRUE, algorithm = NULL)
```

Arguments

`seg`	Vector of segments
`vert`	Vector of vertices
`rivers`	The river network object to use
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.

`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

Distance (numeric)

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

Examples

```
data(Gulk)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

mouthdist(seg=4, vert=40, rivers=Gulk)
mouthdist(seg=c(4,5), vert=c(40,20), rivers=Gulk)
```

`mouthdistbysurvey` *Distance From Mouth for All Observations of Individuals*

Description

Calculates distance from the mouth of a river network to all observations of each individual (given as segment and vertex). and the specified mouth of the river network. The mouth must first be specified (see setmouth). Returns a matrix of distances, with a row for each unique individual and a column for each survey.

A plotting method is provided for the output; see plotseq.

Usage

```
mouthdistbysurvey(
  unique,
  survey,
  seg,
  vert,
  rivers,
  logical = NULL,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment)
`vert`	A vector pf rover coordinates (vertex)
`rivers`	The river network object to use
`logical`	A boolean vector that can be used for subsetting - if used, `mouthdistbysurvey()` will only return distances in which a specified condition is met.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A vector of river network distances (numeric), with each row corresponding to a unique fish and each column corresponding to a unique survey. Values of `NA` indicate the individual not being located during the survey in question.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

plotseq

Examples

```
data(Gulk, fakefish)

seqbysurvey <- mouthdistbysurvey(unique=fakefish$fish.id, survey=fakefish$flight.date,
    seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)
seqbysurvey
plotseq(seqbysurvey)
```

pdist *Pythagorean Distance*

Description

Pythagorean distance between two points. Called internally.

Usage

```
pdist(p1, p2)
```

Arguments

p1	X-Y coordinates of point 1
p2	X-Y coordinates of point 2

Value

Distance (numeric)

Author(s)

Matt Tyers

Examples

```
point1 <- c(1,3)
point2 <- c(4,7)

pdist(point1,point2)
```

pdisttot *Total Pythagorean Distance*

Description

Total Pythagorean distance of a sequence of points. Called internally.

Usage

```
pdisttot(xy)
```

Arguments

xy	A matrix of X-Y coordinates of the sequence of points.

Value

Distance (numeric)

Author(s)

Matt Tyers

Examples

```
points <- matrix(c(1:10), nrow=5, ncol=2, byrow=FALSE)

pdisttot(xy=points)
```

---

plot.homerange *Plot Home Range*

Description

Plotting method for home range, the minimum observed home range for multiple observations of each individual fish.

Usage

```
## S3 method for class 'homerange'
plot(
  x,
  cumulative = FALSE,
  lwd = 3,
  maxlwd = 10,
  col = 4,
  pch = 21,
  label = FALSE,
  main = NULL,
  ...
)
```

Arguments

`x`	An object returned from homerange.
`cumulative`	Whether to plot travel as cumulative, with line thickness depending on the number of times a given region was traveled by a given individual. Defaults to `FALSE`.
`lwd`	The line width for plotting homerange, or minimum line width if `cumulative` is `TRUE`. Defaults to 3.
`maxlwd`	The maximum line width if `cumulative` is `TRUE`. Defaults to 10.
`col`	The line color to use. Defaults to `"blue"`.

`pch`	The point character to use for individual points. Defaults to open circles, the color of lines.
`label`	Whether to add survey labels to individual points, if used in homerange. Defaults to `FALSE`.
`main`	Plot title. If the default `NULL` is used, plots will be titled according to unique individual.
`...`	Additional plotting parameters, see plot.rivernetwork.

Author(s)

Matt Tyers, bug fix by Jordy Bernard

See Also

homerange, homerangeoverlap, plothomerangeoverlap

Examples

```
data(Gulk, fakefish)
ranges <- with(fakefish, homerange(unique=fish.id, survey=flight, seg=seg, vert=vert, rivers=Gulk))
ranges

# 19 plots will be produced, recommend calling par(mfrow=c(4,5))
plot(ranges)
plot(ranges,cumulative=TRUE,label=TRUE)

homerangeoverlap(ranges)

plothomerangeoverlap(ranges)
with(fakefish, riverpoints(seg=seg, vert=vert, rivers=Gulk))
```

`plot.riverdensity` *Plot Kernel Density Using River Distance*

Description

Produces a kernel density plot from a kernel density object created by makeriverdensity.

If the kernel density object includes densities from multiple surveys, a new plot will be created for each survey.

Densities can be displayed using either line widths, color, or both.

The relative densities that are displayed in the plot are calculated according to the form (density/maxdensity)^pwr, with the value of pwr set by the `pwr` argument. Setting `pwr` to a value less than 1 allows smaller values to be more visible on the plot.

Usage

```
## S3 method for class 'riverdensity'
plot(
  x,
  whichplots = NULL,
  points = TRUE,
  bycol = TRUE,
  bylwd = TRUE,
  maxlwd = 10,
  pwr = 0.7,
  scalebyN = TRUE,
  ramp = "grey",
  lwd = 1,
  linecol = "black",
  denscol = "black",
  alpha = 1,
  dark = 1,
  showN = TRUE,
  main = NULL,
  xlab = "",
  ylab = "",
  add = FALSE,
  scalebar = TRUE,
  ...
)
```

Arguments

`x`	A river density object created by makeriverdensity.
`whichplots`	A vector of plots to produce, if multiple plots are produced. For example, specifying `whichplot=c(2,3,4)` will result in only the second, third, and fourth plots of the sequence being produced. Accepting the default (`NULL`) will result in all plots being produced.
`points`	Whether to add the points used for density calculation. Defaults to `TRUE`.
`bycol`	Whether to use a color ramp to show densities. Defaults to `TRUE`.
`bylwd`	Whether to use line thickness to show densities. Defaults to `TRUE`.
`maxlwd`	The maximum line width to use if `bylwd` is set to `TRUE`. Defaults to 10.
`pwr`	The power to use in the nonlinear transformation calculating the relative density values to be displayed (see above.) Defaults to 0.7.
`scalebyN`	Whether to display relative density values scaled by sample size. Specifying `scalebyN=TRUE` will show larger density values associated with surveys with more points, and may be more appropriate for displaying total density. Specifying `scalebyN=FALSE` will allow surveys with smaller sample sizes to be plotted with similar density values as those with larger sample sizes, and may be more appropriate for displaying relative density. Defaults to `TRUE`.

`ramp`	The color ramp used to display densities if `bycol` is set to `TRUE`. Allowed values are `"grey"` (or `"gray"`), `"red"`, `"green"`, `"blue"`, `"heat"`, `"stoplight"`, and `"rainbow"`. Defaults to `"grey"`.
`lwd`	The line width to use for background lines if `bylwd` is set to `TRUE`, or all lines if `bylwd` is set to `FALSE`. Defaults to 1.
`linecol`	The line color to use for background lines if `bylwd` is set to `TRUE`. Defaults to `"black"`.
`denscol`	The line color to use for showing density if `bylwd` is set to `TRUE`. Defaults to `"black"`.
`alpha`	The opacity value for lines. This could potentially allow multiple density plots to be overlayed with different colors.
`dark`	A color-saturation adjustment, with values in [0,1]. A value of 1 uses the true colors, and a value less than 1 will render the colors as slightly darker (less saturated), which may be appear better. Defaults to 1.
`showN`	Whether to automatically include the number of points used as part of the plot title(s).
`main`	Plot title(s), either given as a single text string which is repeated if multiple plots are produced, or a vector of text strings (one for each plot produced). If multiple plots are produced (resulting from multiple surveys), accepting the default (`NULL`) will result in each unique value of survey being used as the plot title, along with the sample size if `showN` is set to `TRUE`.
`xlab`	X-axis label
`ylab`	Y-axis label
`add`	Whether to produce a new plot (`FALSE`), or add to an existing plot (`TRUE`). Defaults to `FALSE`.
`scalebar`	Whether to add a scale bar to plot(s). Defaults to `TRUE`.
`...`	Additional plotting parameters.

Author(s)

Matt Tyers

See Also

makeriverdensity, plotriverdensitypoints

Examples

```
data(Gulk, fakefish)

Gulk_dens <- makeriverdensity(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk,
  survey=fakefish$flight.date)

# # 10 plots will be created, recommend calling par(mfrow=c(2,5))
plot(x=Gulk_dens)
```

`plot.rivernetwork` *Plotting a River Network*

Description

S3 plotting method for the rivernetwork-class. Produces a map of all river segments of a river network object.

Usage

```
## S3 method for class 'rivernetwork'
plot(
  x,
  segmentnum = TRUE,
  offset = TRUE,
  lwd = 1,
  cex = 0.6,
  scale = TRUE,
  color = TRUE,
  empty = FALSE,
  linecol = 1,
  xlab = "",
  ylab = "",
  ...
)
```

Arguments

`x`	The river network object to plot
`segmentnum`	Whether or not to plot segment numbers (defaults to TRUE)
`offset`	Whether to offset segment numbers from lines (defaults to TRUE)
`lwd`	Line width
`cex`	Global character expansion factor for plotting
`scale`	Whether or not to give x- and y-axes the same scale
`color`	How to differentiate segments. If `color==TRUE` (default), segments will be drawn in solid lines with differing colors. If `color==FALSE`, segments will be drawn in the same color with differing line types.
`empty`	Creates an empty plot if set to `TRUE`. Suppresses differentiation by line type if `color==FALSE`, and suppresses segment number labels. Defaults to `FALSE`.
`linecol`	Line color to use if `empty` is `TRUE` or `color` is `FALSE`. Defaults to black.
`xlab`	Label for X-axis (defaults to "")
`ylab`	Label for Y-axis (defaults to "")
`...`	Additional plotting arguments (see par)

Note

This function is intended to provide basic visual checks for the user, not for any real mapping.

Author(s)

Matt Tyers

Examples

```
data(Gulk)
plot(x=Gulk)
```

`plothomerangeoverlap` *Plot Home Range Overlap*

Description

Produces a plot of the overlap of the minimum observed home range for multiple observations of each individual fish, with line thickness illustrating the respective number of individuals' home-ranges represented.

Usage

```
plothomerangeoverlap(x, lwd = 3, maxlwd = 10, col = 4, ...)
```

Arguments

`x`	An object returned from homerange.
`lwd`	Minimum line width to use, defaults to 3.
`maxlwd`	Maximum line width to use, defaults to 10.
`col`	Line color to use, defaults to `"blue"`.
`...`	Additional plotting parameters, see plot.rivernetwork.

Author(s)

Matt Tyers

See Also

homerange, plot.homerange, homerangeoverlap

Examples

```
data(Gulk, fakefish)
ranges <- with(fakefish, homerange(unique=fish.id, survey=flight, seg=seg, vert=vert, rivers=Gulk))
ranges

# 19 plots will be produced, recommend calling par(mfrow=c(4,5))
plot(ranges)
plot(ranges,cumulative=TRUE,label=TRUE)

homerangeoverlap(ranges)

plothomerangeoverlap(ranges)
with(fakefish, riverpoints(seg=seg, vert=vert, rivers=Gulk))
```

`plotmatbysurveylist` *Plot Upstream Distance Between Observations of All Individuals*

Description

Produces a matrix of plots (boxplots are default), with plot `[i,j]` giving the distribution of upstream distances from observation `i` to observation `j`, for all individuals.

Usage

```
plotmatbysurveylist(matbysurveylist, type = "boxplot", showN = TRUE, ...)
```

Arguments

`matbysurveylist`	A list of distance matrices returned from matbysurveylist.
`type`	If `type` is set to `"boxplot"`, boxplots will be produced for each cell. If `type` is set to `"confint"`, lines denoting an approximate 95 percent confidence interval for the mean will be produced instead. If `type` is set to `"dotplot"`, a jittered dotplot will be produced for each cell, which will be the most appropriate if sample sizes are small. Defaults to `"boxplot"`.
`showN`	Whether to display the sample size for each cell. Defaults to TRUE.
`...`	Additional plotting arguments.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

upstream, upstreammatbysurvey

Examples

```
data(Gulk, smallset)
matbysurveylist <- matbysurveylist(unique=smallset$id, survey=smallset$flight, seg=smallset$seg,
   vert=smallset$vert, rivers=Gulk)
plotmatbysurveylist(matbysurveylist)
plotmatbysurveylist(matbysurveylist,type="confint")
plotmatbysurveylist(matbysurveylist,type="dotplot")

data(fakefish)
# matbysurveylist <- matbysurveylist(unique=fakefish$fish.id, survey=fakefish$flight,
#   seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)
# plotmatbysurveylist(matbysurveylist)
```

plotriverdensitypoints

Plot Points Used for Kernel Density

Description

Plots the points used to calculate a kernel density object in makeriverdensity.

This function is intended as a visual check that a sufficient resolution was used.

Usage

```
plotriverdensitypoints(riverdensity)
```

Arguments

`riverdensity`	A river density object created by makeriverdensity.

Author(s)

Matt Tyers

See Also

makeriverdensity, plot.riverdensity

Examples

```
data(Gulk, fakefish)

Gulk_dens <- makeriverdensity(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

plotriverdensitypoints(riverdensity=Gulk_dens)
```

`plotseq` *Plot Sequence of Observations*

Description

Plots the sequence of observations or movements of each individual (given as segment and vertex). This function is primarily intended for use with mouthdistbysurvey, but will also work with riverdistanceseq and upstreamseq.

Usage

```
plotseq(
  seqbysurvey,
  type = "boxplot",
  xlab = "",
  ylab = "",
  main = "",
  cex.axisX = 0.8,
  lowerbound = NULL,
  upperbound = NULL,
  boundtype = "negative",
  surveysareDates = F,
  ...
)
```

Arguments

`seqbysurvey`	A matrix returned from mouthdistbysurvey, riverdistanceseq, or upstreamseq.
`type`	The type of plot to generate. Options are `"boxplot"`,`"dotplot"`,`"boxline"`,or `"dotline"`. Defaults to `"boxplot"`.
`xlab`	X-axis label
`ylab`	Y-axis label
`main`	Plot title
`cex.axisX`	Character expansion factor for X-axis labels
`lowerbound`	An optional vector of lower survey bounds
`upperbound`	An optional vector of upper survey bounds
`boundtype`	Method of plotting survey bounds. Options are `"positive"`, `"negative"` (default), and `"lines"`.
`surveysareDates`	If surveys are in Date format (see as.Date), a value of `TRUE` allows the x-coordinates points to be spaced apart according to date, not equidistantly. Defaults to `FALSE`. Any formatting of the survey variable must be done within the original call to mouthdistbysurvey, riverdistanceseq, or upstreamseq. Dates must already be formatted as dates, or in the form `"YYYY-MM-DD"` or `"YYYY/MM/DD"`.
`...`	Additional plotting parameters

Note

Plots are intended as descriptive only. Any ANOVA-like inference that is suggested from these plots is strongly discouraged. The user is instead advised to use a mixed-effects model or some other inferential tool that accounts for repeated-measures and/or temporal autocorrelation.

Author(s)

Matt Tyers

Examples

```
data(Gulk, fakefish)

x <- mouthdistbysurvey(unique=fakefish$fish.id, survey=fakefish$flight.date,
    seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

plotseq(seqbysurvey=x)
plotseq(seqbysurvey=x, type="boxline")
plotseq(seqbysurvey=x, type="dotplot")
plotseq(seqbysurvey=x, type="dotline")

plotseq(seqbysurvey=x, type="dotline", surveysareDates=TRUE)

from_upstreamseq <- upstreamseq(unique=fakefish$fish.id,
   survey=fakefish$flight, seg=fakefish$seg, vert=fakefish$vert,
   rivers=Gulk)
plotseq(seqbysurvey=from_upstreamseq)
```

---

pointshp2segvert *Convert a Point Shapefile to River Locations*

Description

This function reads a point shapefile and determines the closest vertex in the river network to each point of XY data, returning a data frame with river locations, defined as segment numbers and vertex numbers, along with the data table read from the input shapefile.

Usage

```
pointshp2segvert(path = ".", layer, rivers)
```

Arguments

`path`	File path, default is the current working directory.
`layer`	Name of the shapefile, without the .shp extension.
`rivers`	The river network object to use.

Value

A data frame of river locations, with segment numbers in `$seg`, vertex numbers in `$vert`, snapping distances in `$snapdist`, and the remaining columns corresponding to the data table in the input point shapefile.

Note

If the input shapefile is detected to be in a different projection than the river network, the input shapefile will be re-projected before conversion to river locations.

Author(s)

Matt Tyers

Examples

```
filepath <- system.file("extdata", package="riverdist")

fakefish_UTM5 <- pointshp2segvert(path=filepath, layer="fakefish_UTM5", rivers=Gulk)
head(fakefish_UTM5)

plot(x=Gulk)
points(fakefish_UTM5$x, fakefish_UTM5$y)
riverpoints(seg=fakefish_UTM5$seg, vert=fakefish_UTM5$vert, rivers=Gulk, pch=16, col=2)
```

`removeduplicates` *Remove Duplicates*

Description

Removes duplicated line segments, which can sometimes exist within a shapefile.

Usage

```
removeduplicates(rivers)
```

Arguments

`rivers`	The river network object to use

Value

A new river network object with duplicated segments removed, see rivernetwork

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(abstreams0)
zoomtoseg(seg=c(170,171,157),rivers=abstreams0)

abstreams1 <- removeduplicates(rivers=abstreams0)
zoomtoseg(seg=c(166,167,154),rivers=abstreams1)
```

`removemicrosegs` *Remove Segments that are Smaller than the Connectivity Tolerance*

Description

Automatically detects and removes segments with total displacement (straight-line distance between endpoints) less than the connectivity tolerance. These segments do not serve any real purpose, are bypassed in routes, and cannot be dissolved.

Usage

```
removemicrosegs(rivers)
```

Arguments

`rivers`	The river network object to use.

Value

A new river network object with the specified segments connected (see rivernetwork)

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(abstreams0)
abstreams1 <- removemicrosegs(abstreams0)
```

`removeunconnected` *Remove Unconnected Segments*

Description

Detects and removes segments that are not connected to the river mouth.

Usage

```
removeunconnected(rivers)
```

Arguments

`rivers`	The river network object to use.

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

Examples

```
data(Koyukuk2)
Koy_subset <- trimriver(trimto=c(30,28,29,3,19,27,4),rivers=Koyukuk2)
Koy_subset <- setmouth(seg=1,vert=427,rivers=Koy_subset)
plot(Koy_subset)

Koy_subset_trim <- removeunconnected(Koy_subset)
plot(Koy_subset_trim)
```

`riverdensity` *The "riverdensity" Class*

Description

A class that holds density information computed from point data along a river network.

Details

Created by makeriverdensity from point data and a river network. Contains all information for plotting in plot.riverdensity.

Elements

`densities`**:** Object of class `"list"`. Each list element corresponds to a unique value of survey. Each element is itself of class `"list"`, with each element corresponding to a segment from the associated river network. Each element is a vector of class `"numeric"`, with values equal to the scaled densities calculated at the river network vertices stored in `$densverts` of the associated river network segment.

`endptverts`**:** List of vectors of class `"numeric"`. Each list element is a vector of the vertices of the endpoints of the subsegments considered for density calculation. Each list element corresponds to a river segment from the associated river network.

`densverts`**:** List of vectors of class `"numeric"`. Each element is a vector of the vertices of the points of the subsegments considered for density calculation, that were used for density calculation. Each list element corresponds to a river segment from the associated river network.

`pointsegs`**:** Vector of class `"numeric"`. Defined as the segment numbers of the point data used for density calculation.

`pointverts`**:** Vector of class `"numeric"`. Defined as the vertex numbers of the point data used for density calculation.

`survey`**:** Vector of class `"numeric"` or class `"character"`. Defined as the survey identifiers associated with the point data used for density calculation.

`rivers`**:** Object of class `"rivernetwork"` ; see rivernetwork-class.

Author(s)

Matt Tyers

`riverdirection` *River Direction*

Description

Calculates direction of travel between two points. Only works if river mouth (lowest point) has been specified (see setmouth).

Usage

```
riverdirection(
  startseg,
  endseg,
  startvert,
  endvert,
  rivers,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`startseg`	Segment number of the start of the route
`endseg`	Segment number of the end of the route
`startvert`	Vertex number of the start of the route
`endvert`	Vertex number of the end of the route
`rivers`	The river network object to use
`flowconnected`	If `TRUE`, only returns direction if the two input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

Direction: "up", "down", or "0" (character). Returns NA if `flowconnected==TRUE` and the two segments are not flow-connected.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

setmouth

Examples

```
data(Gulk)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

plot(x=Gulk)
riverdirection(startseg=6, endseg=3, startvert=40, endvert=40, rivers=Gulk)
```

`riverdirectionmat` *River Direction Matrix*

Description

Returns a matrix of calculated travel direction between every point and every other point of given river locations (segment and vertex), or of a subset. The mouth (lowest point) segment and vertex must be specified (see setmouth).

Usage

```
riverdirectionmat(
  seg,
  vert,
  rivers,
  logical = NULL,
  ID = NULL,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use
`logical`	A boolean vector that can be used for subsetting - if used, `riverdirectionmat()` will only return pairwise distances in which a specified condition is met.
`ID`	a vector of observation IDs for aid in interpreting the output table
`flowconnected`	If `TRUE`, only returns direction if the input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of directions (character) with rows and columns labeled by corresponding values of `ID`. See riverdirection for additional information.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdirection

Examples

```
data(Gulk, fakefish)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

logi1 <- (fakefish$flight.date==as.Date("2015-11-25"))

riverdirectionmat(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, logical=logi1)
```

`riverdirectionmatbysurvey` *River Direction Matrix of All Observations of an Individual*

Description

Returns a matrix of travel direction between all observations of one unique fish.

Usage

```
riverdirectionmatbysurvey(
  indiv,
  unique,
  survey,
  seg,
  vert,
  rivers,
  full = TRUE,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`indiv`	The unique identifier of the fish in question.
`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`full`	Whether to return the full matrix, with `NA` values for missing data (`TRUE`), or a the subset of rows and columns corresponding to successful observations. Defaults to `TRUE`.
`flowconnected`	If `TRUE`, only returns direction if the input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of directions (character), with rows and columns defined by survey. In the resulting matrix, the element with the row identified as `A` and column identified as `B` is defined as the direction traveled from survey A to survey B. Therefore, it is likely that only the upper triangle of the matrix will be of interest.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdirection

Examples

```
data(Gulk, fakefish)
riverdirectionmatbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

riverdirectionmatbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, full=FALSE)
```

riverdirectionseq *River Travel Direction Between Sequential Observations*

Description

Returns a matrix of directions traveled by unique fish between sequential surveys. The mouth (lowest point) segment and vertex must be specified (see setmouth).

Usage

```
riverdirectionseq(
  unique,
  survey,
  seg,
  vert,
  rivers,
  logical = NULL,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`logical`	A boolean vector that can be used for subsetting - if used, `riverdirectionseq()` will only return pairwise distances in which a specified condition is met.
`flowconnected`	If `TRUE`, only returns direction if the input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A data frame of directions (character), with rows defined by unique fish and columns defined by observation increment (1 to 2, 2 to 3, etc.) See riverdirection for additional information.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdirection

Examples

```
data(Gulk, fakefish)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

riverdirectionseq(unique=fakefish$fish.id, survey=fakefish$flight, seg=fakefish$seg,
   vert=fakefish$vert, rivers=Gulk)

riverdirectionseq(unique=fakefish$fish.id, survey=fakefish$flight.date, seg=fakefish$seg,
   vert=fakefish$vert, rivers=Gulk)
```

`riverdirectiontofrom` *River Direction Matrix between Two Datasets*

Description

Returns a matrix of directions between each river location in two datasets, with one expressed as rows and the other expressed as columns.

Usage

```
riverdirectiontofrom(
  seg1,
  vert1,
  seg2,
  vert2,
  rivers,
  logical1 = NULL,
  logical2 = NULL,
  ID1 = NULL,
  ID2 = NULL,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg1`	First vector of river locations (segment component). These are expressed as rows in the output matrix.
`vert1`	First vector of river locations (vertex component). These are expressed as rows in the output matrix.
`seg2`	Second vector of river locations (segment component). These are expressed as columns in the output matrix.
`vert2`	Second vector of river locations (vertex component). These are expressed as columns in the output matrix.
`rivers`	The river network object to use.
`logical1`	A boolean vector that can be used for subsetting. If used, `riverdirectiontofrom` will only return directions in which a specified condition is met for the first dataset.
`logical2`	A boolean vector that can be used for subsetting. If used, `riverdirectiontofrom` will only return directions in which a specified condition is met for the second dataset.
`ID1`	a vector of observation IDs for the first dataset that will be used as row names in the output matrix.
`ID2`	a vector of observation IDs for the second dataset that will be used as column names in the output matrix.
`flowconnected`	If `TRUE`, only returns distance if the input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of directions (character) with rows and columns labeled by corresponding values of `ID`. See riverdirection for additional information.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdirection

Examples

```
data(Gulk)

streamlocs.seg <- c(1,8,11)
streamlocs.vert <- c(50,70,90)
streamlocs.ID <- c("A","B","C")

fish.seg <- c(1,4,9,12,14)
fish.vert <- c(10,11,12,13,14)
fish.ID <- c("fish1","fish2","fish3","fish4","fish5")

Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)

riverdirectiontofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk,
  ID1=streamlocs.ID, ID2=fish.ID)

logi1 <- streamlocs.ID=="B" | streamlocs.ID=="C"
logi2 <- fish.ID!="fish3"

riverdirectiontofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk, logical1=logi1,
  logical2=logi2, ID1=streamlocs.ID, ID2=fish.ID)
```

`riverdistance` *River Distance*

Description

Calculates the total river network distance between two points on the river network, given in river locations (segment and vertex).

Usage

```
riverdistance(
  startseg = NULL,
  endseg = NULL,
  startvert,
  endvert,
  rivers,
  path = NULL,
  map = FALSE,
  add = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`startseg`	Segment number of the start of the route
`endseg`	Segment number of the end of the route
`startvert`	Vertex number of the start of the route
`endvert`	Vertex number of the end of the route
`rivers`	The river network object to use
`path`	(optional) The vector-format route of segment numbers can also be supplied instead of the starting and ending segments.
`map`	Whether or not to draw a sanity-check map, showing the calculated route in entirety. Defaults to FALSE.
`add`	If `map==TRUE`, whether to add the route drawing to an existing plot (`add=TRUE`) or produce a new plot (`add=FALSE`).
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, `riverdistance()` will return `NA`. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

Total route distance, in the units of the coordinate system used (this will likely be meters).

Note

If a distance lookup table (`$distlookup`) is present in the river network object, accepting `NULL` will bypass route detection and return distance automatically, the fastest algorithm of all. This is done automatically in buildsegroutes, but can be called directly using buildlookup.

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

Examples

```
data(Gulk)
riverdistance(startseg=6, endseg=14, startvert=100, endvert=200, rivers=Gulk)
riverdistance(startvert=100, endvert=200, path=c(6,3,4,10,11,14), rivers=Gulk)
riverdistance(startseg=6, endseg=14, startvert=100, endvert=200, rivers=Gulk, map=TRUE)

# speed comparison:

data(abstreams)
```

```
tstart <- Sys.time()
riverdistance(startseg=120, startvert=10, endseg=131, endvert=10, rivers=abstreams,
              algorithm="sequential")
Sys.time()- tstart

tstart <- Sys.time()
riverdistance(startseg=120, startvert=10, endseg=131, endvert=10, rivers=abstreams,
              algorithm="Dijkstra")
Sys.time()- tstart

tstart <- Sys.time()
riverdistance(startseg=120, startvert=10, endseg=131, endvert=10, rivers=abstreams)

# Note: it is not necessary to specify the algorithm here: the distance function
# will automatically select the fastest algorithm unless otherwise specified.
Sys.time()- tstart
```

`riverdistancelist` *Multiple River Distances*

Description

Used to calculate a list of possible river distances, in the event of braiding. Calls routelist to detect a list of routes from one river location to another, and uses riverdistance to calculate the distances along those routes. Different routes are detected by randomly reordering the segment numbers of the input river network object, thus changing the internal hierarchy of segment selection.

Usage

```
riverdistancelist(startseg, endseg, startvert, endvert, rivers, reps = 100)
```

Arguments

`startseg`	Segment number of the start of the route
`endseg`	Segment number of the end of the route
`startvert`	Vertex number of the start of the route
`endvert`	Vertex number of the end of the route
`rivers`	The river network object to use
`reps`	Deprecated. Was the number of randomized reorderings to try.

Value

A list with two objects, `$routes` being a list of detected routes in ascending order by distance, and `$distances` being the respective distances along the routes detected.

Note

Since this function uses randomization, there is no guarantee that the list of routes will be comprehensive. Larger numbers of reps can be tried, but computation can be slow, particularly in the presence of a complex river network. It may be advantageous to use trimriver to create a smaller, more specific river network object to work with.

Author(s)

Matt Tyers

Examples

```
data(KilleyW)
plot(x=KilleyW)

Killey.dists <- riverdistancelist(startseg=1, endseg=16, startvert=100, endvert=25,
   rivers=KilleyW, reps=1000)
Killey.dists  # 18 routes are detected.

# mapping the shortest route detected...
riverdistance(startvert=100, endvert=25, path=Killey.dists$routes[[1]], rivers=KilleyW, map=TRUE)

# mapping the shortest longest detected...
riverdistance(startvert=100, endvert=25, path=Killey.dists$routes[[18]], rivers=KilleyW, map=TRUE)
```

---

riverdistancemat *River Distance Matrix*

Description

Returns a matrix of distances between every point and every other point of given river locations (segment and vertex), or of a subset.

Usage

```
riverdistancemat(
  seg,
  vert,
  rivers,
  logical = NULL,
  ID = NULL,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`logical`	A boolean vector that can be used for subsetting. If used, `riverdistancemat` will only return pairwise distances in which a specified condition is met.
`ID`	a vector of observation IDs for aid in interpreting the output table
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of distances (numeric) with rows and columns labeled by corresponding values of `ID`.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdistance

Examples

```
data(Gulk, fakefish)

logi1 <- (fakefish$flight.date==as.Date("2015-11-25"))

riverdistancemat(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, logical=logi1)
```

riverdistancematbysurvey

River Distance Matrix of All Observations of an Individual

Description

Returns a matrix of network distances between all observations of one unique fish.

Usage

```
riverdistancematbysurvey(
  indiv,
  unique,
  survey,
  seg,
  vert,
  rivers,
  full = TRUE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`indiv`	The unique identifier of the fish in question.
`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`full`	Whether to return the full matrix, with `NA` values for missing data (`TRUE`), or a the subset of rows and columns corresponding to successful observations. Defaults to `TRUE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of distances (numeric), with rows and columns defined by survey.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdistance

Examples

```
data(Gulk, fakefish)
riverdistancematbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

riverdistancematbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, full=FALSE)
```

`riverdistanceseq` *River Distance Between Sequential Observations*

Description

Returns a matrix of distances traveled by unique fish, between sequential surveys. A plotting method is also provided for the output; see plotseq

Usage

```
riverdistanceseq(
  unique,
  survey,
  seg,
  vert,
  rivers,
  logical = NULL,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.

`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`logical`	A boolean vector that can be used for subsetting. If used, `riverdistanceseq()` will only return pairwise distances in which a specified condition is met.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A data frame of distances (numeric), with rows defined by unique fish and columns defined by observation increment (1 to 2, 2 to 3, etc.)

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdistance, plotseq

Examples

```
data(Gulk, fakefish)
riverdistanceseq(unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

seqbysurvey <- riverdistanceseq(unique=fakefish$fish.id, survey=fakefish$flight.date,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)
seqbysurvey
plotseq(seqbysurvey)
```

`riverdistancetofrom` *River Distance Matrix between Two Datasets*

Description

Returns a matrix of distances between each river location in two datasets, with one expressed as rows and the other expressed as columns.

Usage

```
riverdistancetofrom(
  seg1,
  vert1,
  seg2,
  vert2,
  rivers,
  logical1 = NULL,
  logical2 = NULL,
  ID1 = NULL,
  ID2 = NULL,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg1`	First vector of river locations (segment component). These are expressed as rows in the output matrix.
`vert1`	First vector of river locations (vertex component). These are expressed as rows in the output matrix.
`seg2`	Second vector of river locations (segment component). These are expressed as columns in the output matrix.
`vert2`	Second vector of river locations (vertex component). These are expressed as columns in the output matrix.
`rivers`	The river network object to use.
`logical1`	A boolean vector that can be used for subsetting. If used, `riverdistancetofrom` will only return distances in which a specified condition is met for the first dataset.
`logical2`	A boolean vector that can be used for subsetting. If used, `riverdistancetofrom` will only return distances in which a specified condition is met for the second dataset.
`ID1`	a vector of observation IDs for the first dataset that will be used as row names in the output matrix.
`ID2`	a vector of observation IDs for the second dataset that will be used as column names in the output matrix.

`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of distances (numeric) with rows and columns labeled by corresponding values of `ID`.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

riverdistance

Examples

```
data(Gulk)

streamlocs.seg <- c(1,8,11)
streamlocs.vert <- c(50,70,90)
streamlocs.ID <- c("A","B","C")

fish.seg <- c(1,4,9,12,14)
fish.vert <- c(10,11,12,13,14)
fish.ID <- c("fish1","fish2","fish3","fish4","fish5")

riverdistancetofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk, ID1=streamlocs.ID, ID2=fish.ID)

logi1 <- streamlocs.ID=="B" | streamlocs.ID=="C"
logi2 <- fish.ID!="fish3"

riverdistancetofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk, logical1=logi1, logical2=logi2,
  ID1=streamlocs.ID, ID2=fish.ID)
```

`rivernetwork` *The "rivernetwork" Class*

Description

A class that holds spatial coordinates for river networks, as well as network topology and attributes.

Details

Created by line2network from an input line shapefile. Contains all information for network distance calculation, plotting, etc. in the 'riverdist' package.

Plotting methods are described in plot.rivernetwork.

Elements

`sp`: Object of class `"SpatialLinesDataFrame"` from package 'sp'; see SpatialLinesDataFrame-class. This is the original object as read by readOGR, and is preserved to maintain plotting capability.

`lines`: Object of class `"list"`. Each list element is a matrix of XY coordinates of the vertices of a single river segment.

`connections`: Object of class `"matrix"`, with `"numeric"` elements. Defined as a square matrix, with elements describing the type of connection detected between line segments.

- A value of 1 in element `[i,j]` indicates that the beginning of segment `i` is connected to the beginning of segment `j`.
- A value of 2 in element `[i,j]` indicates that the beginning of segment `i` is connected to the end of segment `j`.
- A value of 3 in element `[i,j]` indicates that the end of segment `i` is connected to the beginning of segment `j`.
- A value of 4 in element `[i,j]` indicates that the end of segment `i` is connected to the end of segment `j`.
- A value of 5 in element `[i,j]` indicates that segments `i` and `j` are connected at both beginning and end.
- A value of 6 in element `[i,j]` indicates that the beginning of segment `i` is connected to the end of segment `j`, and the end of segment `i` is connected to the beginning of segment `j`.
- A value of NA in element `[i,j]` indicates that segments `i` and `j` are not connected.

`lengths`: Vector of class `"numeric"`. Defined as the calculated total lengths of each river segment.

`names`: Vector of class `"character"`. Defined as the names of each river segment.

`mouth`: Object of class `"list"`, with two elements. Element `mouth.seg` gives the segment number of the mouth (lowest point) of the river network, and `mouth.vert` gives the vertex number.

`sequenced`: `"logical"`: has value of TRUE if line vertices have been stored in upstream sequence using sequenceverts.

`tolerance`: `"numeric"`: the spatial tolerance that was used in determining river segment endpoint connectivity; see line2network, splitsegments.

`units`: `"character"`: the spatial units detected from the input shapefile.

`lineID`: Object of class `"data.frame"` establishing the relationship between river segments as stored in the `sp` and `lines` elements, and is used for updating the `sp` element during river network editing in dissolve, splitsegments, sequenceverts, trimriver, and trimtopoints.

- `rivID` gives the list element number of each river segment in `lines`. This is the same number that is used for segment numbering in river locations.
- `sp_line` gives the corresponding list element in `sp@lines`.
- `sp_seg` gives the corresponding list element in `sp@lines[[]]@Lines`.

`braided`: `"logical"`: Has value of `TRUE` if checkbraidedTF has detected braiding, `FALSE` if no braiding has been detected, and `NA` if braiding has not yet been checked.

`cumuldist`: List of class `"numeric"`: Each element is a vector of cumulative distances along each river segment, beginning with 0.

`segroutes`: Object of class `"list"`, with each element defined as a vector of class `"numeric"`, describing the route from the mouth segment to the specific segment. This element only exists if buildsegroutes has been run, and can greatly speed up route and distance calculation.

`distlookup`: List of three matrices, of class `"numeric"` or `"logical"`. Element `[i,j]` of each matrix corresponds to the route between segment `i` and `j`. The `distlookup$middist` matrix gives the total distance of the "middle" of each route (between the starting and ending segments"), and the `distlookup$starttop` and `distlookup$endtop` matrices have value `TRUE`, `FALSE`, or `NA` if the segments at the beginning or end of the route are connected to the rest of the route at the top of the coordinate matrix, bottom of the coordinate matrix, or if the route is contained to just one segment, respectively.

Author(s)

Matt Tyers

`riverpoints` *Draw Points from River Locations*

Description

Adds points to an active plot. Works like points but with river locations (segments and vertices) rather than xy coordinates.

Usage

```
riverpoints(seg, vert, rivers, pch = 1, col = 1, jitter = 0, ...)
```

Arguments

`seg`	A vector of segments
`vert`	A vector of vertices
`rivers`	The river network object to use

`pch`	Point character, as a vector or single value
`col`	Point color, as a vector or single value
`jitter`	Maximum amount of random noise to add to "jitter" points if desired, so points do not overlap one another
`...`	Additional arguments for points

Author(s)

Matt Tyers

Examples

```
data(fakefish,Gulk)

plot(x=Gulk, xlim=c(862000,882000), ylim=c(6978000,6993000))
points(x=fakefish$x, y=fakefish$y, pch=16, col=2)
riverpoints(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, pch=15, col=4)

plot(x=Gulk, empty=TRUE)
with(fakefish, riverpoints(seg=seg, vert=vert, rivers=Gulk,
       pch=16, col=flight, jitter=1000))
```

`routelist` *Detect Multiple Routes*

Description

Called internally within riverdistancelist. Detects all possible routes from one river network segment to another, in the event of braiding.

Usage

```
routelist(startseg, endseg, rivers, reps = 100)
```

Arguments

`startseg`	Segment number of the start of the route
`endseg`	Segment number of the end of the route
`rivers`	The river network object to use
`reps`	Deprecated. Was used in a previous version using randomization.

Value

A list of vectors, each describing a route in segment numbers.

Note

The previous version of this function returned many possible routes using randomization - this algorithm now computes all possible routes.

Author(s)

Matt Tyers

Examples

```
data(KilleyW)
plot(x=KilleyW)

routelist(startseg=1, endseg=16, rivers=KilleyW, reps=1000)
```

`sequenceverts` *Store Vertices in Ascending Sequence*

Description

Rearranges the vertices of a river network object so that vertices are stored sequentially moving up river for all segments (coordinates [1,] are the bottom of each segment).

Usage

```
sequenceverts(rivers)
```

Arguments

`rivers`	The river network object to use

Value

A new river network object (see rivernetwork)

Note

Even without calling `sequenceverts`, the vertices will be stored sequentially - either moving up river or down for a given segment. What `sequenceverts()` adds is a standardized direction.

Currently, no function in package 'riverdist' requires the vertices to be stored sequentially.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Gulk)
Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)
str(Gulk)

Gulk.dir <- sequenceverts(rivers=Gulk)
str(Gulk.dir)
```

---

setmouth — *Specify the Segment and Vertex of the Mouth of a River Network Object.*

Description

Provides a user-friendly way of specifying the segment and vertex of the mouth (lowest point) of a river network object.

Usage

```
setmouth(seg, vert, rivers)
```

Arguments

`seg`	The segment number to store for the mouth
`vert`	The vertex number to store for the mouth
`rivers`	The river network object to use

Value

A new river network object (see rivernetwork)

Note

The mouth segment and vertex can also be specified using direct assignment to the `$mouth$seg` and `$mouth$vert` components of the river network object.

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Gulk)

# say we know that segment 1 is the lowest segment in this river network, but we don't know
# which end is the mouth.
showends(seg=1, rivers=Gulk)

# this means that the mouth is row 1, so we can specify this:
Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)
```

---

showends *Identify Vertex Coordinates of Segment Endpoints*

Description

Identifies the vertex coordinates (row numbers) of the endpoints of a given segment. The main purpose is determining which of the endpoints is the mouth (or lowest point) of the river system.

Usage

```
showends(seg, rivers)
```

Arguments

`seg`	The segment (number) to check
`rivers`	The river network object to use

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

Examples

```
data(Gulk)

# say we know that segment 1 is the lowest segment in this river network, but we don't know
# which end is the mouth.
showends(seg=1, rivers=Gulk)

# this means that the mouth is row 1, so we can specify this:
Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)
```

smallset *Dataset: Smallset*

Description

A small set of observations of fakefish on the Gulkana River and its tributaries.

Usage

```
data(smallset)
```

Format

A data frame

Details

- `x`. X-coordinate of observation (Alaska Albers Equal Area). Note that the locations do not align with the river network object.
- `y`. Y-coordinate of observation
- `seg`. River segment
- `vert`. River vertex
- `fish.id`. Numeric identifier for each fish (individual fish were observed more than once)
- `flight`. Numeric identifier for each telemetry flight

See Also

Gulk

splitsegmentat *Split a Segment at a Specified Vertex*

Description

Splits a segment at a specified vertex, creating two new segments.

Usage

```
splitsegmentat(seg, vert, rivers)
```

Arguments

`seg`	The segment to split
`vert`	The vertex to split it at
`rivers`	The river network object to use

Value

A new, updated river network object

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Gulk)
plot(x=Gulk)

Gulk2 <- splitsegmentat(seg=1, vert=400, rivers=Gulk)
plot(x=Gulk2)
```

`splitsegments` *Split Segments by Endpoint Proximity*

Description

Detects cases in which segments should be split to establish appropriate topology, and splits them. Specifically, it looks for segment endpoints intersecting (or within a tolerance of) another segment. It then splits the intersected segment at the point where the endpoint of the other segment breaks it.

Usage

```
splitsegments(
  rivers,
  tolerance = NULL,
  splitthese = NULL,
  splitthemat = NULL,
  one2one = FALSE,
  append = FALSE
)
```

Arguments

`rivers`	The river network object to use
`tolerance`	The spatial snapping tolerance to use for detecting intersection. If a NULL value is used (default), it will default to the tolerance that was used in river network creation in line2network.
`splitthese`	An optional vector of target segments to split. If this argument is used, only these segments will be split. If the default (`NULL`) is accepted, all segments will be used.

`splitthemat`	An optional vector of segments (endpoints) to use for splitting. If this argument is used, segments will only be split at the endpoints of these segments. If the default (`NULL`) is accepted, all segments will be used.
`one2one`	Logical, indicating a one-to-one correspondence between arguments `splitthese` and `splitthemat`. Defaults to `FALSE`,
`append`	Logical, indicating how to organize the output river network. If `TRUE`, appends newly-created segments to the end of `$lines`, rather than retaining original line ordering. This may be useful in retaining original line ID. Defaults to `FALSE`.

Value

A new, updated river network object

Note

This function is called within cleanup, which is recommended in most cases.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Koyukuk1)
topologydots(rivers=Koyukuk1)
# Segments 7, 8, 13, and 16 need to be split so topologies will work.
# Since endpoints are not in the same place, they are not detected as
# being connected.
plot(x=Koyukuk1)

Koyukuk1split <- splitsegments(rivers=Koyukuk1)
topologydots(rivers=Koyukuk1split)
plot(x=Koyukuk1split)

# if only segment 17 were to be split in three places
plot(x=splitsegments(rivers=Koyukuk1, splitthese=c(7,7,7),
        splitthemat=c(14,5,12)))

# if only segment 16 were to be split, showing behavior of append=
plot(x=splitsegments(rivers=Koyukuk1, splitthese=c(7,7,7),
        splitthemat=c(14,5,12), append=TRUE))
```

topologydots *Check Connectivity of a River Network Object*

Description

Produces a graphical check of the connectivity of a river network object. It produces a plot of the river network object, and overlays red dots at non-connected endpoints and green dots at connected endpoints.

Usage

```
topologydots(rivers, add = FALSE, ...)
```

Arguments

`rivers`	The river network object to check
`add`	Whether call a new plot (`FALSE`) or add dots to an existing plot (`TRUE`). Defaults to `FALSE`.
`...`	Additional plotting arguments (see par)

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Gulk)
topologydots(rivers=Gulk)
```

trimriver *Trim a River Network Object to Specified Segments*

Description

Removes line segments from a river network object. User can specify which segments to remove (`trim`) or which segments to keep (`trimto`).

Usage

```
trimriver(trim = NULL, trimto = NULL, rivers)
```

Arguments

`trim`	Vector of line segments to remove
`trimto`	Vector of line segments to keep
`rivers`	The river network object

Value

A new river network object

Note

Specifying segments in both trim and trimto arguments will result in an error.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Kenai1)
plot(x=Kenai1)

Kenai1.trim <- trimriver(trim=c(46,32,115,174,169,114,124,142,80), rivers=Kenai1)
plot(x=Kenai1.trim)

Kenai1.trim.2 <- trimriver(trimto=c(20,57,118,183,45,162,39,98,19), rivers=Kenai1)
plot(x=Kenai1.trim.2)
```

`trimtopoints` *Trim a River Network to a Set of X-Y Coordinates*

Description

Removes line segments from a river network object that are not adjacent to a set of point data, given in X-Y coordinates.

Usage

```
trimtopoints(x, y, rivers, method = "snap", dist = NULL)
```

Arguments

`x`	Vector of x-coordinates of point data to buffer around
`y`	Vector of y-coordinates of point data to buffer around
`rivers`	The river network object to use
`method`	Three methods are available. If `"snap"` is specified (the default), only the closest segment to each point is retained. If `"snaproute"` is specified, segments are also retained that will maintain total connectivity in the resulting river network. If `"buffer"` is specified, all segments with endpoints or midpoints within `dist` units of the input locations are retained.
`dist`	Distance to use for buffering, if `method="buffer"`. If this is not specified, the maximum spread in the x- and y- direction will be used.

Value

A new river network object (see rivernetwork)

Note

If `method=="buffer"`, only distances to segment endpoints and midpoints are checked, and still only whole segments are removed.

Author(s)

Matt Tyers

See Also

line2network

Examples

```
data(Koyukuk2)
x <- c(139241.0, 139416.1, 124600.1, 122226.8)
y <- c(1917577, 1913864, 1898723, 1898792)

plot(x=Koyukuk2)
points(x, y, pch=15, col=4)
legend(par("usr")[1], par("usr")[4], legend="points to buffer around", pch=15, col=4, cex=.6)

Koyukuk2.buf1 <- trimtopoints(x, y, rivers=Koyukuk2, method="snap")
plot(x=Koyukuk2.buf1)
points(x, y, pch=15, col=4)

Koyukuk2.buf2 <- trimtopoints(x, y, rivers=Koyukuk2, method="snaproute")
plot(x=Koyukuk2.buf2)
points(x, y, pch=15, col=4)

Koyukuk2.buf3 <- trimtopoints(x, y, rivers=Koyukuk2, method="buffer", dist=1000)
plot(x=Koyukuk2.buf3)
points(x, y, pch=15, col=4)
```

`upstream` *Upstream River Distance*

Description

Calculates river network distances as +/-, defined as upriver/downriver.

Specifying `net=TRUE` will compute net upriver distance (3 river km down a tributary and then 15 river km up the mainstem will mean 12 rkm net. Otherwise the function will return 18 rkm upriver travel.)

The mouth (lowest point) segment and vertex must be specified (see setmouth).

Usage

```
upstream(
  startseg,
  endseg,
  startvert,
  endvert,
  rivers,
  flowconnected = FALSE,
  net = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`startseg`	Segment number of the start of the route
`endseg`	Segment number of the end of the route
`startvert`	Vertex number of the start of the route
`endvert`	Vertex number of the end of the route
`rivers`	The river network object to use
`flowconnected`	If `TRUE`, only returns distance if the two input segments are flow-connected. Defaults to `FALSE`.
`net`	Whether to calculate net distance (`net=TRUE`) or total distance (`net=FALSE`)
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

Upstream distance (numeric). Returns NA if `flowconnected` has value `TRUE` and the two segments are not flow-connected.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

setmouth

Examples

```
data(Gulk)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

plot(x=Gulk)
riverpoints(seg=c(6,4), vert=c(140,140), pch=16, col=2, rivers=Gulk)
upstream(startseg=6, endseg=4, startvert=140, endvert=40, rivers=Gulk, net=TRUE)
upstream(startseg=6, endseg=4, startvert=140, endvert=40, rivers=Gulk, net=FALSE)
upstream(startseg=6, endseg=4, startvert=140, endvert=40, rivers=Gulk, flowconnected=TRUE)
```

---

upstreammat *Upstream Distance Matrix*

Description

Returns a matrix of upstream distance between every point and every other point of given river locations (segment and vertex), or of a subset. The mouth (lowest point) segment and vertex must be specified (see setmouth).

Usage

```
upstreammat(
  seg,
  vert,
  rivers,
  logical = NULL,
  ID = NULL,
  flowconnected = FALSE,
  net = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`logical`	A boolean vector that can be used for subsetting - if used, riverdirectionseq() will only return pairwise distances in which a specified condition is met.
`ID`	a vector of observation IDs for aid in interpreting the output table
`flowconnected`	If `TRUE`, only returns distance if the input segments are flow-connected. Defaults to `FALSE`.
`net`	Whether to calculate net upstream distance (net=TRUE) or total distance (net=FALSE, default). See upstream.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of upstream distances (numeric) with rows and columns labeled by corresponding values of `ID`. See upstream for additional information.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

upstream

Examples

```
data(Gulk, fakefish)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

logi1 <- (fakefish$flight.date==as.Date("2015-11-25"))

upstreammat(seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, logical=logi1)
```

`upstreammatbysurvey` *Upstream Distance Matrix of All Observations of an Individual*

Description

Returns a matrix of upstream travel distance between all observations of one unique fish.

Usage

```
upstreammatbysurvey(
  indiv,
  unique,
  survey,
  seg,
  vert,
  rivers,
  full = TRUE,
  flowconnected = FALSE,
  net = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`indiv`	The unique identifier of the fish in question.
`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`full`	Whether to return the full matrix, with `NA` values for missing data (`TRUE`), or a the subset of rows and columns corresponding to successful observations. Defaults to `TRUE`.

`flowconnected`	If `TRUE`, only returns direction if the input segments are flow-connected. Defaults to `FALSE`.
`net`	Whether to calculate net upstream distance (net=TRUE) or total distance (net=FALSE, default).
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of upstream distances (numeric), with rows and columns defined by survey. In the resulting matrix, the element with the row identified as `A` and column identified as `B` is defined as the upstream distance traveled from survey A to survey B. Therefore, it is likely that only the upper triangle of the matrix will be of interest.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

upstream

Examples

```
data(Gulk, fakefish)
upstreammatbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk)

upstreammatbysurvey(indiv=1, unique=fakefish$fish.id, survey=fakefish$flight,
      seg=fakefish$seg, vert=fakefish$vert, rivers=Gulk, full=FALSE)
```

`upstreamseq` *Upstream Distance Between Sequential Observations*

Description

Returns a matrix of distance with direction by unique fish between sequential surveys. The mouth (lowest point) segment and vertex must be specified (see setmouth). A plotting method is provided for the output; see plotseq.

Usage

```
upstreamseq(
  unique,
  survey,
  seg,
  vert,
  rivers,
  logical = NULL,
  flowconnected = FALSE,
  net = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`unique`	A vector of identifiers for each fish.
`survey`	A vector of identifiers for each survey. It is recommended to use a numeric or date format (see as.Date) to preserve survey order.
`seg`	A vector of river locations (segment component).
`vert`	A vector of river locations (vertex component).
`rivers`	The river network object to use.
`logical`	A boolean vector that can be used for subsetting - if used, `upstreamseq()` will only return pairwise distances in which a specified condition is met.
`flowconnected`	If `TRUE`, only returns distance if the input segments are flow-connected. Defaults to `FALSE`.
`net`	Whether to calculate net upstream distance (net=TRUE) or total distance (net=FALSE, default).
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A data frame of upstream distances (numeric), with rows defined by unique fish and columns defined by observation increment (1 to 2, 2 to 3, etc.) See upstream for additional information.

Note

Returns either net upstream distance (net=TRUE) or total distance (net=FALSE, default). See upstream.

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

upstream, plotseq

Examples

```
data(Gulk, fakefish)

# Mouth must be specified
Gulk$mouth$mouth.seg <- 1
Gulk$mouth$mouth.vert <- 1

upstreamseq(unique=fakefish$fish.id, survey=fakefish$flight, seg=fakefish$seg,
      vert=fakefish$vert, rivers=Gulk)

seqbysurvey <- upstreamseq(unique=fakefish$fish.id, survey=fakefish$flight.date, seg=fakefish$seg,
      vert=fakefish$vert, rivers=Gulk)
seqbysurvey
plotseq(seqbysurvey)
```

upstreamtofrom *Upstream Distance Matrix between Two Datasets*

Description

Returns a matrix of upstream distances between each river location in two datasets, with one expressed as rows and the other expressed as columns.

Usage

```
upstreamtofrom(
  seg1,
  vert1,
  seg2,
  vert2,
  rivers,
  logical1 = NULL,
  logical2 = NULL,
  ID1 = NULL,
  ID2 = NULL,
  net = FALSE,
  flowconnected = FALSE,
  stopiferror = TRUE,
  algorithm = NULL
)
```

Arguments

`seg1`	First vector of river locations (segment component). These are expressed as rows in the output matrix.
`vert1`	First vector of river locations (vertex component). These are expressed as rows in the output matrix.
`seg2`	Second vector of river locations (segment component). These are expressed as columns in the output matrix.
`vert2`	Second vector of river locations (vertex component). These are expressed as columns in the output matrix.
`rivers`	The river network object to use.
`logical1`	A boolean vector that can be used for subsetting. If used, `upstreamtofrom` will only return upstream distances in which a specified condition is met for the first dataset.
`logical2`	A boolean vector that can be used for subsetting. If used, `upstreamtofrom` will only return upstream distances in which a specified condition is met for the second dataset.
`ID1`	a vector of observation IDs for the first dataset that will be used as row names in the output matrix.
`ID2`	a vector of observation IDs for the second dataset that will be used as column names in the output matrix.
`net`	Whether to calculate net upstream distance (`TRUE`) or signed total distance (`FALSE`). See upstream.
`flowconnected`	If `TRUE`, only returns distance if the input segments are flow-connected. Defaults to `FALSE`.
`stopiferror`	Whether or not to exit with an error if a route cannot be found. If this is set to `FALSE` and a route cannot be found, the function will return `NA` in the appropriate entry. Defaults to `TRUE`. See detectroute.
`algorithm`	Which route detection algorithm to use (`"Dijkstra"`, `"sequential"`, or `"segroutes"`). If left as `NULL` (the default), the function will automatically make a selection. See detectroute for more details.

Value

A matrix of upstream distances (numeric) with rows and columns labeled by corresponding values of `ID`. See upstream for additional information.

Note

Building routes from the river mouth to each river network segment and/or distance lookup tables will greatly reduce computation time (see buildsegroutes).

Author(s)

Matt Tyers

See Also

upstream

Examples

```
data(Gulk)

streamlocs.seg <- c(1,8,11)
streamlocs.vert <- c(50,70,90)
streamlocs.ID <- c("A","B","C")

fish.seg <- c(1,4,9,12,14)
fish.vert <- c(10,11,12,13,14)
fish.ID <- c("fish1","fish2","fish3","fish4","fish5")

Gulk <- setmouth(seg=1, vert=1, rivers=Gulk)

upstreamtofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk,
  ID1=streamlocs.ID, ID2=fish.ID)

logi1 <- streamlocs.ID=="B" | streamlocs.ID=="C"
logi2 <- fish.ID!="fish3"

upstreamtofrom(seg1=streamlocs.seg, vert1=streamlocs.vert,
  seg2=fish.seg, vert2=fish.vert, rivers=Gulk, logical1=logi1,
  logical2=logi2, ID1=streamlocs.ID, ID2=fish.ID)
```

---

whoconnected *Check Which Segments are Connected to a Given Segment.*

Description

Returns which segments are connected to a specified segment within a river network. It may be useful for error checking.

Usage

```
whoconnected(seg, rivers)
```

Arguments

seg	The segment to check
rivers	The river network object it belongs to

Value

A vector of segment numbers

Author(s)

Matt Tyers

Examples

```
data(Gulk)
plot(Gulk)
whoconnected(seg=4, rivers=Gulk)
```

`xy2segvert` *Convert XY Coordinates to River Locations*

Description

This function determines the closest vertex in the river network to each point of XY data and returns a list of river locations, defined as segment numbers and vertex numbers.

Usage

```
xy2segvert(x, y, rivers)
```

Arguments

`x`	A vector of x-coordinates to transform
`y`	A vector of y-coordinates to transform
`rivers`	The river network object to use

Value

A data frame of river locations, with segment numbers in `$seg`, vertex numbers in `$vert`, and the snapping distance for each point in `$snapdist`.

Note

Conversion to river locations is only valid if the input XY coordinates and river network are in the same projected coordinate system. Point data in geographic coordinates can be projected using project in package 'rgdal', and an example is shown below.

Author(s)

Matt Tyers

Examples

```
data(Gulk,fakefish)
head(fakefish)

fakefish.riv <- xy2segvert(x=fakefish$x, y=fakefish$y, rivers=Gulk)
head(fakefish.riv)

plot(x=Gulk, xlim=c(862000,882000), ylim=c(6978000,6993000))
points(fakefish$x, fakefish$y, pch=16, col=2)
riverpoints(seg=fakefish.riv$seg, vert=fakefish.riv$vert, rivers=Gulk, pch=15, col=4)


## converting a matrix of points stored in long-lat to Alaska Albers Equal Area:
data(line98, Kenai1)
head(line98)  # note that coordinates are stored in long-lat, NOT lat-long

library(rgdal)
line98albers <- project(line98,proj="+proj=aea +lat_1=55 +lat_2=65
    +lat_0=50 +lon_0=-154 +x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs
    +ellps=GRS80")
head(line98albers)

zoomtoseg(seg=c(162,19), rivers=Kenai1)
points(line98albers)
```

---

zoomtoseg *Zoom to segment*

Description

Calls plot.rivernetwork and automatically zooms to a specified segment or vector of segments. Not intended for any real mapping - just investigating and error checking.

Usage

```
zoomtoseg(seg, rivers, ...)
```

Arguments

seg	A segment or vector of segments to zoom to
rivers	The river network object to use
...	Additional plotting arguments (see par)

Author(s)

Matt Tyers

Examples

```
data(Kenai3)
plot(x=Kenai3)

# checking out a particularly messy region...
zoomtoseg(c(110,63), rivers=Kenai3)
```

Index

∗ **datasets**
    abstreams, 5
    abstreams0, 6
    fakefish, 19
    fakefish_density, 19
    Gulk, 20
    Kenai1, 25
    Kenai2, 25
    Kenai3, 26
    KilleyW, 28
    Koyukuk0, 29
    Koyukuk1, 29
    Koyukuk2, 30
    line98, 32
    smallset, 77

abstreams, 5
abstreams0, 6
addcumuldist, 6
addverts, 7, *13*
as.Date, *35*, *38*, *48*, *57*, *58*, *66*, *67*, *86*, *88*

buildlookup, 8, *9*, *28*, *33*, *62*
buildsegroutes, *8*, 9, *13*, *17*, *22*, *28*, *33*, *36–38*, *46*, *54*, *56*, *57*, *59*, *60*, *62*, *65*, *67*, *68*, *70*, *72*, *84*, *85*, *87*, *88*, *90*

calculateconnections, 10
checkbraided, 11
checkbraidedTF, 12, *13*, *72*
cleanup, *7–9*, *11*, *12*, 13, *15*, *18*, *51*, *52*, *75*, *76*, *79*
cleanup_verts, 14
connectsegs, *13*, 15

detectroute, 16, *21*, *24*, *36–38*, *54*, *55*, *57*, *58*, *60*, *62*, *65*, *66*, *68*, *70*, *83*, *85*, *87*, *88*, *90*
dissolve, *13*, 18, *72*

fakefish, 19
fakefish_density, 19

Gulk, *19*, 20, *77*

highlightseg, 20
homerange, 21, *22*, *23*, *40*, *41*, *45*
homerange-class, *22*, 22
homerangeoverlap, *22*, 23, *41*, *45*

isflowconnected, 24

Kenai1, *25*, 25, *26*
Kenai2, *25*, 25, *26*
Kenai3, *25*, 26, *28*
kfunc, 26
KilleyW, 28
Koyukuk0, 29
Koyukuk1, *29*, 29, *30*
Koyukuk2, *29*, 30

line2network, *29*, 30, *71*, *74*, *75*, *78–82*
line98, 32

makeriverdensity, 32, *41–43*, *47*, *52*
mapbyname, 34
mapriver (plot.rivernetwork), 44
matbysurveylist, 34, *46*
mouthdist, 36
mouthdistbysurvey, 37, *48*

par, *20*, *34*, *44*, *80*, *93*
pdist, 39
pdisttot, 39
plot, *80*
plot.homerange, *21–23*, 40, *45*
plot.riverdensity, *32*, *33*, 41, *47*, *52*
plot.rivernetwork, *41*, 44, *45*, *71*, *93*
plothomerangeoverlap, *22*, *23*, *41*, 45
plotmatbysurveylist, *34*, *36*, 46
plotriverdensity, *19*, *20*
plotriverdensity (plot.riverdensity), 41

plotriverdensitypoints, *33*, *43*, 47
plotseq, *37, 38*, 48, *67, 68*, *87*, *89*
points, *72, 73*
pointshp2segvert, 49
project, *92*

readOGR, *30*, *71*
removeduplicates, *13*, 50
removemicrosegs, *13*, 51
removeunconnected, *13*, 52
riverdensity, *19*, 52
riverdensity-class, *19*, *33*
riverdensity-class *(*riverdensity*)*, 52
riverdirection, *24*, *36*, 53, *55–60*
riverdirectionmat, 55
riverdirectionmatbysurvey, *35, 36*, 56
riverdirectionseq, 58
riverdirectiontofrom, 59
riverdist *(*riverdist-package*)*, 3
riverdist-package, 3
riverdistance, *16*, *36*, 61, *63*, *65*, *67, 68*, *70*
riverdistancelist, 63, *73*
riverdistancemat, 64
riverdistancematbysurvey, *35, 36*, 66
riverdistanceseq, *48*, 67
riverdistancetofrom, 69
rivernetwork, *5–9*, *13–15*, *20*, *25, 26*, *28–30*, *50, 51*, 71, *74, 75*, *82*
rivernetwork-class, *6*, *10*, *31*, *44*, *53*
rivernetwork-class *(*rivernetwork*)*, 71
riverpoints, 72
routelist, *63*, 73

sequenceverts, *71, 72*, 74
setmouth, *9*, *13*, *24*, *36, 37*, *53–55*, *58*, 75, *83, 84*, *87*
showends, 76
smallset, 77
SpatialLinesDataFrame-class, *71*
splitsegmentat, 77
splitsegments, *13*, *71, 72*, 78
spTransform, *31*

topologydots, 80
trimriver, *64*, *72*, 80
trimtopoints, *72*, 81

upstream, *24*, *36*, *47*, 83, *85*, *87–91*
upstreammat, 84
upstreammatbysurvey, *35, 36*, *47*, 86
upstreamseq, *48*, 87
upstreamtofrom, 89

whoconnected, 91

xy2segvert, 92

zoomtoseg, 93