Package 'rivnet'

July 23, 2025

Type Package

Title Extract and Analyze Rivers from Elevation Data

Version 0.6.0

Description Seamless extraction of river networks from digital elevation models data. The package allows analysis of digital elevation models that can be either externally provided or downloaded from open source repositories (thus interfacing with the 'elevatr' package). Extraction is performed via the 'D8' flow direction algorithm of TauDEM (Terrain Analysis Using Digital Elevation Models), thus interfacing with the 'traudem' package. Resulting river networks are compatible with functions from the 'OCNet' package. See Carraro (2023) <doi:10.5194/hess-27-3733-2023> for a presentation of the package.

Imports spam, raster, sf, terra, traudem (>= 1.0.3), elevatr, OCNet (>= 1.1.0), methods, Rcpp (>= 1.0.9), curl, fields, parallelly (>= 1.33.0)

License MIT + file LICENSE

Encoding UTF-8

Suggests knitr, rmarkdown, bookdown

VignetteBuilder knitr

URL https://lucarraro.github.io/rivnet/

BugReports https://github.com/lucarraro/rivnet/issues

LinkingTo Rcpp

NeedsCompilation yes

Author Luca Carraro [cre, aut], University of Zurich [cph, fnd]

Maintainer Luca Carraro <luca.carraro@hotmail.it>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2025-02-12 13:20:02 UTC

Contents

`rivnet-package` *Extract and Analyze Rivers from Elevation Data.*

Description

Seamless extraction of river networks from digital elevation models data. The package allows analysis of digital elevation models that can be either externally provided or downloaded from open source repositories (thus interfacing with the `elevatr` package). Extraction is performed via the 'D8' flow direction algorithm of TauDEM (Terrain Analysis Using Digital Elevation Models), thus interfacing with the `traudem` package. Resulting river networks are compatible with functions from the `OCNet` package.

Author(s)

Luca Carraro (<luca.carraro@hotmail.it>)

aggregate_river *Aggregate a river*

Description

Aggregates a river

Usage

```
aggregate_river(river, ...)
```

Arguments

river	A river object.
...	Further arguments to be passed to aggregate_OCN.

Details

This is an alias to OCNet::aggregate_OCN.

Value

A river object. See aggregate_OCN for description of its structure.

Examples

```
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478,237413),
DEM=fp)
r <- aggregate_river(r)
```

contour_to_shapefile *Export catchment contour as shapefile*

Description

Export catchment contour as shapefile.

Usage

```
contour_to_shapefile(river, filename,
                     EPSG = NULL, ...)
```

Arguments

river	river object.
filename	Character. Output file name. It must contain the ".shp" extension.
EPSG	EPSG code. Default is NULL, which does not produce a .prj file (i.e., the shapefile does not contain projection information).
...	Additional arguments to be passed to writeVector (e.g., overwrite = TRUE allows overwriting an existing shapefile).

Value

No output is produced. This function is used for its side effetcs.

See Also

terra::writeVector

Examples

```
library(terra) # to use "vect"
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413), DEM=fp)

tmpname <- paste0(tempfile(), ".shp")
contour_to_shapefile(river, tmpname, overwrite = TRUE)

# read output
vv <- vect(tmpname)
vv
plot(vv)


# export contour shapefile for multiple catchments
river <- extract_river(outlet=data.frame(x=c(637478,629532),y=c(237413,233782)),
                  EPSG=21781, #CH1903/LV03 coordinate system
                  ext=c(6.2e5,6.6e5,2e5,2.5e5),
                  z=8)

contour_to_shapefile(river, tmpname, overwrite = TRUE)
vv <- vect(tmpname)
vv
plot(vv)

# add projection
contour_to_shapefile(river, tmpname,
EPSG = 21781,
overwrite = TRUE)
vv <- vect(tmpname)
vv
```

covariate_river *Attribute covariates to nodes of a river network*

Description

Attributes covariate values from raster files to subcatchments of a river object. Both local and upstream-averaged covariate values are calculated.

Usage

```
covariate_river(x, river, categorical = TRUE, overwrite = FALSE)
```

Arguments

x	SpatRaster object (obtained via rast) containing categorical or continuous variables from which covariates are computed. Its coordinate system must be the same of river. Consider using terra::project to change coordinate system.
river	river object. It must have been aggregated (via aggregate_river).
categorical	Logical. Is the covariate categorical (e.g. land cover classes)? If x consists of several layers, it is possible to specify categorical as a vector of logicals (one per each layer); alternatively, the single value of categorical is applied to all layers of x.
overwrite	Logical. If TRUE, overwrite previously calculated covariates.

Details

If categorical = TRUE, the number of columns of SC$locCov, SC$upsCov is equal to the number of unique values of x within the catchment. Column names are composed as "y_z", where y = names(x) and z are the unique values of x. Values correspond to the fraction of pixels (FD nodes) within the local/upstream area that are covered by a given category (e.g., land cover type).

If categorical = FALSE, SC$locCov and SC$upsCov have a single column named names(x). Values correspond to the mean covariate value within the local/upstream reference area.

If x has multiple layers, columns in the data frames are added sequentially. The same occurs if covariate_river is run repeated times (for instance, to compute covariates for one SpatRaster object at a time) when overwrite = FALSE.

Value

A river object. The following elements are added:

SC$locCov	Data frame of covariates evaluated as local values for each subcatchment (i.e., mean covariate value within a catchment).
SC$upsCov	Data frame of covariates evaluated as upstream-averaged values for each subcatchment (i.e., mean covariate value within the area upstream of a given subcatchment, including the subcatchment itself).

See Also

terra::rast, terra::project

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413),
                       DEM=fp)
river <- aggregate_river(river)

# land cover raster file (categorical)
r1 <- terra::rast(system.file("extdata/landcover.tif", package="rivnet"))
# legend: 1-urban; 2-agriculture; 3-forest; 4-improductive

river <- covariate_river(r1, river)

plot(river$SC$locCov[ , 1], river) # fraction of urban area within a subcatchment
plot(river$SC$upsCov[ , 1], river) # fraction of upstream-averaged urban area

# mean air temperature raster file (continuous)
r2 <- terra::rast(system.file("extdata/temperature.tif", package="rivnet"))

river <- covariate_river(r2, river, categorical = FALSE)

plot(river$SC$locCov[, 5], river) # the layer has been added after the 4 previous ones

names(river$SC$locCov)
```

---

extract_river *Extract a river*

Description

Function that extracts a river network from elevation data via TauDEM's D8 flow direction algorithm. It can return a river object and/or output from TauDEM functions as a raster file. Elevation data can be either downloaded from the web or provided externally.

Usage

```
extract_river(outlet, EPSG=NULL, ext=NULL, z=NULL, DEM=NULL,
  as.river=TRUE, as.rast=FALSE, filename=NULL, showPlot=FALSE,
  threshold_parameter=1000, n_processes=1, displayUpdates=0, src="aws",
  args_get_elev_raster=list())
```

Arguments

outlet	A vector, matrix or data frame expressing the coordinates of the river outlet(s) (in the coordinate system identified by EPSG, or the same as in DEM, if the latter is provided). If a vector, the odd components identify the longitudinal (x) coordinates, and the even components the latitudinal (y) coordinates. If a matrix, it should have 2 columns (for x and y coordinates respectively) and number of rows equal to the number of outlets. If a data frame, it should have components outlet$x, outlet$y identifying the respective coordinates.
EPSG	EPSG code identifying the coordinate system used. See https://epsg.org/. It is required if DEM is not specified, and not used otherwise. It is recommended to use projected coordinate systems, so that lengths and areas in the river object are in metric units.
ext	Vector expressing the extent of the region where elevation data are downloaded. It must be in the form c(xmin, xmax, ymin, ymax). Coordinates must be expressed in the coordinate system identified by EPSG. It is required if DEM is not specified, and not used otherwise.
z	Zoom level at which elevation data should be downloaded. See get_elev_raster for details. It is required if DEM is not specified, and not used otherwise.
DEM	Filename of the Digital Elevation Model raster file to be used.
as.river	Logical. Should a river object be created?
as.rast	Logical. Should a raster file containing results from traudem functions output be created?
filename	Filename of the raster file produced if as.rast=TRUE. Only required if as.rast=TRUE. It can be a single filename, or four filenames, in which case four different raster files are produced. See Details.
showPlot	Logical. Should a plot of the calculated contributing area and extracted catchment contour be produced?
threshold_parameter	Value passed to taudem_threshold. See example and 'traudem' documentation for details.
n_processes	Value passed to the traudem functions.
displayUpdates	Numeric. Possible values are 0, 1, 2. If 0, console output is suppressed (barring some messages from elevatr::get_elev_raster, if DEM is not provided). If 1, succint console output is produced. If 2, extensive console output from elevatr::get_elev_raster and traudem functions is displayed.
src	Value passed to get_elev_raster. Deprecated.
args_get_elev_raster	List of additional parameters to be passed to get_elev_raster. Parameters included in this list override default options (for example, if two different zoom values are specified in args_get_elev_raster$z and z, then the latter is not considered.)

Details

This is a wrapper to `elevatr` and `traudem` functions, allowing a seamless extraction of river networks from elevation data. The output `river` object is compatible with `OCNet` functions (it is equivalent to an OCN produced by `landscape_OCN`).

The workflow of TauDEM commands used is as follows: `PitRemove` -> `D8FlowDir` -> `D8ContributingArea` -> `StreamDefByThreshold` -> `MoveOutletsToStreams` -> `D8ContributingArea`. See https://hydrology.usu.edu/taudem/taudem5/index.html for details on TauDEM.

When `as.rast = TRUE`, a raster file is returned. It consists of four layers:

fel pit-filled elevation data

p D8 flow directions

ad8 contributing area for the whole region

ssa contributing area with respect to the outlet(s) used

The raster file is written via `terra::writeRaster`.

If nested outlets are specified, the function ignores the upstream outlet.

Value

A `river` object. See `create_OCN`, `landscape_OCN` for description of its structure.

See Also

`elevatr::get_elev_raster`, `traudem::taudem_threshold`, `OCNet::create_OCN`, `OCNet::landscape_OCN`.

Examples

```
# extract the river Wigger (Switzerland) from DEM raster file
# outlet coordinates are expressed in the CH1903/LV03 coordinate system
# (i.e. same as the DEM file)
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478,237413),
DEM=fp)
r



# same as above but download DEM data via elevatr
r <- extract_river(outlet=c(637478,237413),
EPSG=21781, #CH1903/LV03 coordinate system
ext=c(6.2e5,6.6e5,2e5,2.5e5),
z=8)



# enhance resolution by increasing zoom
r2 <- extract_river(outlet=c(637478,237413),
EPSG=21781, #CH1903/LV03 coordinate system
ext=c(6.2e5,6.6e5,2e5,2.5e5),
```

```
z=9)
plot(r)
plot(r2)



# specify two outlets as a data frame
r <- extract_river(outlet=data.frame(x=c(637478,629532),y=c(237413,233782)),
                   EPSG=21781, #CH1903/LV03 coordinate system
                   ext=c(6.2e5,6.6e5,2e5,2.5e5),
                   z=10, showPlot=TRUE)
plot(r)

r <- aggregate_river(r)
plot(r, chooseCM = 2)  # display only the second catchment
# (i.e. that identified by the second outlet)



# effect of threshold_parameter
r <- extract_river(outlet = c(637478, 237413),
                   EPSG = 21781, #CH1903/LV03 coordinate system
                   ext = c(6.2e5, 6.6e5, 2e5, 2.5e5),
                   z = 8, threshold_parameter = 50,
    showPlot = TRUE)
plot(r) # if threshold_parameter is too small, the outlet might be located
# in a smaller river reach, and the extracted river network would be too small
# showPlot = TRUE can help identify what is going on

r <- extract_river(outlet = c(637478, 237413),
                   EPSG = 21781, #CH1903/LV03 coordinate system
                   ext = c(6.2e5, 6.6e5, 2e5, 2.5e5),
                   z = 8, threshold_parameter = 1e5,
    showPlot = TRUE)
plot(r) # if threshold_parameter is too large, the outlet pixel might not be
# located at all (for instance, in this case no cells have contributing area
# above threshold_parameter), hence throwing an error
```

`get_riverweight` *Get raster values at specific river locations*

Description

Get values from a raster at specific locations in a river network. It can be used to extract relevant values from upstream-averaged rasters as produced by `rast_riverweight`.

Usage

```
get_riverweight(x, rst, river, args_locate_site = list())
```

Arguments

x	Coordinate(s) of the location(s) of interest. These can be expressed as a 2-valued vector (indicating longitudinal and latitudinal coordinate of the single point of interest–in the same coordinate system as `rst` and `river`), a matrix or data frame (with two columns, for longitudinal and latitudinal coordinates, respectively, and as many rows as the number of points of interest). See examples.
rst	A `SpatRaster` object as obtained by `rast_riverweight`.
river	A `river` object generated via `aggregate_river`.
args_locate_site	List of arguments to be passed to `locate_site`. See examples.

Value

A data frame with columns named after the layers of `rst`, and as many rows as the number of rows in `x`.

See Also

`rast_riverweight`, `locate_site`

Examples

```
data(wigger)

r1 <- terra::rast(system.file("extdata/landcover.tif", package = "rivnet"))
# legend: 1-urban; 2-agriculture; 3-forest; 4-improductive

r.exp <- rast_riverweight(r1, wigger)

v1 <- get_riverweight(c(641000, 226100), r.exp, wigger) # from vector
m <- matrix(c(641000, 226100), 1, 2)
v2 <- get_riverweight(m, r.exp, wigger) # from matrix
df <- data.frame(m)
v3 <- get_riverweight(df, r.exp, wigger) # from data frame

m2 <- matrix(c(641000, 226100, 639600, 226100), 2, 2, byrow = TRUE)
v4 <- get_riverweight(m2, r.exp , wigger) # from matrix multipoint


# use showPlot = TRUE from locate_site to check snapping of point to the river network
v1 <- get_riverweight(c(641000, 226100), r.exp, wigger,
args_locate_site = list(showPlot = TRUE))
```

hydro_river *Assign hydraulic variables to a river network*

Description

Assign hydraulic variables (width, water depth, discharge, water velocity, ...) across a `river` object from measured values based on scaling relationships and/or uniform flow equations.

Usage

```
hydro_river(x, river, level = "AG", leopold = TRUE,
 expWidth = 0.5, expDepth = 0.4, expQ = 1,
 crossSection = "natural", ks = 30, minSlope = NULL)
```

Arguments

`x`	Data frame containing measured hydraulic variables. It must consist of columns `data` (numeric values of width in m, depth in m or discharge in m^3 s^(-1)), `type` (containing the variable type: width (`"w"`), depth (`"d"`) or discharge (`"Q"`)), and `node` (ID of AG or RN nodes to where `data` have been measured). See details.
`river`	`river` object. It must have been aggregated (via `aggregate_river`).
`level`	Aggregation level at which the nodes in `x$node` are defined and at which hydraulic variables are calculated. Possible values are `"RN"`, `"AG"`. See `OCNet::OCNet` for details.
`leopold`	Logical. Should scaling relationships of hydraulic variables with drainage area (in the spirit of Leopold and Maddock, 1953) be preferred over uniform flow (Gauchler-Strickler/Manning) relationships? See details.
`expWidth`	Exponent for the scaling relationship of width to drainage area. See details.
`expDepth`	Exponent for the scaling relationship of depth to drainage area. See details.
`expQ`	Exponent for the scaling relationship of discharge to drainage area. See details.
`crossSection`	Shape of the river cross-section (constant across the river network). Possible values are `"rectangular"`, `"natural"`, or a numeric value indicating the exponent of the width-depth relationship. See details.
`ks`	Roughness coefficient according to Gauchler-Strickler (in m^(1/3)s^(-1)). It is the inverse of Manning's roughness coefficient. It can be a single value (thus assumed constant for the whole river network), or a vector of length equal to the number of nodes at the specified `level` (one roughness coefficient value for each corresponding node).
`minSlope`	Minimum slope value, replacing null or `NaN` values of slope in the `river` object. If `NULL`, it is assumed equal to either `river$slope0` (if the river is an OCN) or to the minimum positive slope value at the selected `level`.

Details

This function is a more complete version of `rivergeometry_OCN`.

`x` must consist of at least one width value and one value of either depth or discharge. All values included in `x$data` must be referred to the same time point, so that spatial interpolation can be performed. If the goal is assessing spatio-temporal changes in hydraulic variables, then `hydro_river` must be run independently for each time point. For each node, one cannot specify multiple values of the same variable type. Function `locate_site` can be used to attribute `x$node`.

If `level = "AG"`, the drainage area values used for the power law relationships are calculated as `0.5*(river$AG$A + river$AG$AReach)`. If `level = "RN"`, `river$RN$A` is used.

Width values in `x` are assumed to be measured at the water surface. If a single width value is provided, widths are calculated at all nodes from a power-law relationship on drainage area with exponent `expWidth` and such that width at the measured node is equal to the provided value. If multiple values of width are provided, the function fits a width-drainage area power law on the provided values. In this case, `expWidth` is not used and the output (fitted) width values at the measured nodes are generally different than the observed ones.

Depending on the type of depth and discharge data in `x`, the function behaves in eight different ways:

1. If one depth value and zero discharge values are provided, the Gauchler-Strickler uniform flow relationship (hereafter GS) is applied to find discharge at the node where depth was measured. Discharge values are then attributed to all nodes based on a power-law relationship vs. drainage area (hereafter PL) with exponent `expQ`. Finally, depth values at all nodes are derived from GS.
2. If one discharge value and zero depth values are provided, discharge values are attributed to all nodes based on a PL with exponent `expQ`, and such that the value at the measurement node be equal to the observed one. Depth values at all nodes are then derived from GS.
3. If one discharge and one depth value are provided (not necessarily referred to the same node), discharge values are first attributed as in case 2. If `leopold = TRUE`, depth values are derived from a PL with exponent `expDepth`; conversely, GS is applied.
4. If multiple values of discharge and zero values of depth are provided, discharge values are attributed from a power-law fit on measured values vs. drainage area (heareafter PLF). Depth values are then obtained from GS.
5. If multiple values of depth and zero values of discharge are provided, depth values are obtained by PLF. Discharge values are then calculated from GS.
6. If multiple values of discharge and one value of depth are provided, discharge values are first computed as in case 4. Depth values are then obtained by either PF with exponent `expDepth` (if `leopold = TRUE`), or alternatively via GS.
7. If multiple values of depth and one value of discharge are provided, depth values are first computed as in case 5. Discharge values are then obtained as in case 2 (if `leopold = TRUE`), or alternatively computed from GS.
8. If multiple values of both discharge and depth are provided, discharge values are computed as in case 4, and depth values are computed as in case 5.

Cross-sections are assumed as vertically symmetric. If `crossSection = "natural"`, the relationship between width and depth at a cross-section is expressed by width ~ depth^0.65, as suggested by

Leopold and Maddock (1953) (where width ~ discharge^0.26 and depth ~ discharge^0.4). Assuming `crossSection = 0` is equivalent to `"rectangular"` (width does not depend on depth), while `crossSection = 1` corresponds to an isosceles triangular cross-section.

Value

A `river` object. The following elements are added to the list indicated by `level`:

`width`, `depth`, `discharge`
: Values assigned at all nodes (see Details). Units are m, m, m^3 s^(-1), respectively.

`velocity`
: Values in m s^(-1). Calculated by continuity (i.e., ratio between discharge and cross-sectional area).

`volume`
: Values in m^3. Calculated as cross-sectional area times length.

`hydraulicRadius`
: Values in m. Calculated as the ratio between cross-sectional area and wetted perimeter.

`shearStress`
: Values in N m^(-2). Shear stress exerted at the streambed by waterflow. Calculated as gamma*hydraulicRadius*slope, where gamma = 9806 N m^(-3) is the specific weight of water.

See Also

aggregate_river, locate_site, OCNet::rivergeometry_OCN.

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413),
                       DEM=fp)
river <- aggregate_river(river)

data <- c(12.8,  6.7,  3.3,  1.1,  9.5,  0.8)
type <- c("w",   "w",  "w",  "w",  "Q",  "d")
node <- c( 46,   109,  181,  145,  46,   46) # assume these have been found via locate_site

x <- data.frame(data=data, type=type, node=node)

river1 <- hydro_river(x, river) # case 3
river2 <- hydro_river(x, river, leopold = FALSE) # case 3 (depth calculated via GS)

plot(0.5*(river1$AG$A + river1$AG$AReach), river1$AG$depth) # Power law with exponent 0.4
plot(0.5*(river2$AG$A + river2$AG$AReach),
river2$AG$depth) # Higher depths in reaches with small slope

river3 <- hydro_river(x, river, leopold = FALSE, minSlope = 0.002)
plot(0.5*(river1$AG$A + river1$AG$AReach), river1$AG$depth) # Variability is reduced
```

```
river <- hydro_river(x[-5, ], river) # case 1
river <- hydro_river(x[-6, ], river) # case 2
```

`locate_site` *Locate site in a river*

Description

Finds location of a site (with coordinates X, Y) within a `river` object.

Usage

```
locate_site(X, Y = NULL, river, euclidean = TRUE, showPlot = FALSE,
  xlim = NULL, ylim = NULL)
```

Arguments

`X`	Either a list or a numeric value. If `X` is a list, it must contain the longitudinal and latitudinal coordinates of the site in columns `x` (or `X`) and `y` (or `Y`), respectively. If numeric, it represents the longitudinal coordinate of the site. Coordinate values must be given in the same coordinate system as the `river` object.
`Y`	Latitudinal coordinate of the site. If `X` is a list, it can contain the `river` object.
`river`	`river` object where the site is to be located. It must have been aggregated (via `aggregate_river`).
`euclidean`	Logical. Should the location of the site in the `river` be attributed "as the crow flies"? If `FALSE`, the downstream path from (`X`, `Y`) is followed until the river network is met.
`showPlot`	Logical. Should a close-up of the relocated site be shown?
`xlim`, `ylim`	Ranges of x- and y-axis limits for the plot. Only active if `showPlot = TRUE`.

Details

This function identifies the node in the river network (at the RN and AG levels) that is closest to an arbitrary site of coordinates `X`, `Y`. Only a single site can be processed per function call.

Desired coordinates X, Y can be found in an interactive way by clicking on the river map and using function `locator`.

Nodes at the RN level thus found can be defined as new breakpoints for reaches (see `aggregate_OCN` and argument `breakpoints`).

Value

A list with objects:

FDode	index at the FD level of the node that is closest to (X, Y). This is generally not a node that belongs to the river network.
distance	The distance between FDnode and RNnode, expressed either as Euclidean distance (if euclidean = TRUE), or as downstream distance alternatively.
RNode	index at the RN level of the relocated site.
AGode	index at the AG level of the relocated site.
par	List of graphical parameters as produced by a call to par (only if showPlot = TRUE). This can be used to add features to the existing plot. Deprecated.

See Also

OCNet::aggregate_OCN, locator

Examples

```
fp <- system.file("extdata/wigger.tif", package = "rivnet")
r <- extract_river(outlet = c(637478, 237413),
                DEM = fp)
r <- aggregate_river(r)

X <- 641329; Y <- 227414
out1 <- locate_site(X, Y, r, showPlot = TRUE) # as the crow flies
out2 <- locate_site(X, Y, r, showPlot = TRUE, euclidean = FALSE) # follow downstream path



 # define X, Y by clicking on the map
if (interactive()) {
fp <- system.file("extdata/wigger.tif", package = "rivnet")
r <- extract_river(outlet = c(637478, 237413),
                DEM = fp)
r <- aggregate_river(r)
plot(r)

point <- locator(1) # click on the map to define point
locate_site(point$X, point$Y, r)

# alternative: specify X as a list and pass river as second argument
locate_site(point, r)
}
```

paths_river *Find paths in a river*

Description

Find paths in a river

Usage

```
paths_river(river, ...)
```

Arguments

river	A river object.
...	Further arguments to be passed to paths_OCN.

Details

This is an alias to OCNet::paths_OCN.

Value

A river object. See paths_OCN for description of its structure.

Examples

```
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478,237413),
DEM=fp)
r <- aggregate_river(r)
r <- paths_river(r)
```

path_velocities_river *Calculate velocities along paths in a river*

Description

Calculate mean water velocities along paths in a river object.

Usage

```
path_velocities_river(river, level = c("RN", "AG"),
  displayUpdates = FALSE)
```

Arguments

river	A river object. It must have been aggregated (via aggregate_river), and contain paths (via paths_river with includeDownstreamNode = TRUE) and velocities (via hydro_river or rivergeometry_OCN) at the desired aggregation level(s). See level.
level	Aggregation level(s) at which path velocities should be calculated. Possible values are "RN", "AG", c("RN","AG").
displayUpdates	Logical. State if updates are printed on the console while path_velocities_river runs.

Details

Velocities are calculated by dividing the total distance (length of the downstream path joining two nodes) by the total time (sum of times taken to cover all nodes in between the origin and destination nodes; such times are calculated as length/velocity).

Note that paths may or may not include the downstream node; this is controlled by option includeDownstreamNode in paths_river. Path velocities are calculated accordingly. In both cases, diagonal entries of pathVelocity are set equal to the respective node velocity. See example.

Value

A river object. The following element is added to the list indicated by level:

pathVelocities	It is a spam object. pathVelocities[i, j] is the mean water velocity along the downstream path from nodes i to j, and is null if the two nodes are not connected by such a path.

See Also

paths_river, hydro_river, OCNet::rivergeometry_OCN.

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413),
                       DEM=fp)
river <- aggregate_river(river)
river <- paths_river(river, includePaths = TRUE)
river <- OCNet::rivergeometry_OCN(river) # simplified alternative to hydro_river
                                  # to attribute velocities at all RN and AG nodes

river <- path_velocities_river(river, level = "AG") # downstream nodes are not included in paths
river$AG$pathVelocities[176, 176]
river$AG$pathVelocities[176, 174]
 # node 174 is immediately downstream of 176; if downstream nodes are not included
 # in paths, the two velocities are equal

river2 <- paths_river(river, includePaths = TRUE, includeDownstreamNode = TRUE)
```

```
river2 <- path_velocities_river(river2, level = "AG") # now downstream nodes are included in paths
river2$AG$pathVelocities[176, 176]
river2$AG$pathVelocities[176, 174]
```

---

plot *Plot a river*

Description

Plots a river object

Usage

```
## S4 method for signature 'river,numeric'
plot(x, y, type, ...)
## S4 method for signature 'numeric,river'
plot(x, y, type, ...)
## S4 method for signature 'river,missing'
plot(x, type, ...)
```

Arguments

x	A river object (or a numeric vector if y is a river).
y	A numeric vector to be displayed (or a river if x is a numeric vector). It is equivalent to theme in draw_subcatchments_OCN and draw_thematic_OCN. If y is specified, the river must have been aggregated. See details.
type	Optional argument. If type = "SC" or type = "subcatchments", draw_subcatchments_OCN is used (provided that the river object is aggregated); if type = "elev2D", draw_elev2D_OCN is used; if type = "contour", draw_contour_OCN is used (provided that the river object contains the CM field as produced by landscape_OCN or extract_river); alternatively, draw_thematic_OCN is used.
...	Arguments passed to the plotting functions draw_simple_OCN, draw_contour_OCN, draw_thematic_OCN. See details.

Details

This is an interface to the plotting functions draw_simple_OCN, draw_elev2D_OCN, draw_contour_OCN, draw_subcatchments_OCN, draw_thematic_OCN. If the river object does not have an elevation field (i.e., it has been generated by create_OCN or create_general_contour_OCN, but landscape_OCN has not been run), the plotting function used is draw_simple_OCN. If the elevation field is present, but the river has not been aggregated (via aggregate_OCN or aggregate_river), the default plotting function used is draw_contour_OCN. If the river has been aggregated, draw_subcatchments_OCN or draw_thematic_OCN are used depending on type. Elevation maps can be produced with type = "elev2D", regardless of whether the river has been aggregated.

Adding scale bar and north arrow. Scale bar and north arrow can be added via `terra`'s functions `terra::sbar` and `terra::north`, respectively. However, note that arguments `d` and `xy` must be specified by the user (because no `rast` object is plotted). See example.

See Also

`OCNet::draw_simple_OCN`, `OCNet::draw_elev2D_OCN`, `OCNet::draw_contour_OCN`, `OCNet::draw_subcatchments_OCN`, `OCNet::draw_thematic_OCN`

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
r <- extract_river(outlet=c(637478,237413),
DEM=fp)
plot(r) # equivalent to draw_contour_OCN


r <- aggregate_river(r)
plot(r) # equivalent to draw_thematic_OCN
plot(r, type = "SC") # equivalent to draw_subcatchments_OCN
plot(r, type = "contour") # equivalent to draw_contour_OCN

# equivalent to draw_thematic_OCN with 'theme' specified
plot(r, r$AG$streamOrder, discreteLevels = TRUE)
plot(r$AG$streamOrder, r, discreteLevels = TRUE)  # swapping arguments is allowed

# equivalent to draw_subcatchments_OCN with 'theme' specified
plot(r, r$SC$Y, type = "SC", addLegend = FALSE)
plot(r$SC$Y, r,  type = "subcatchments", addLegend = FALSE)  # swapping arguments is allowed

# plot elevation map
plot(r, type = "elev2D", drawRiver = TRUE)
# now add scale bar and north arrow
library(terra)
# sbar() # this would throw an error
# north()# this would throw an error
sbar(d=1000, xy=c(min(r$FD$X), min(r$FD$Y)-r$cellsize)) # this works
north(d=1000, xy=c(max(r$FD$X)+r$cellsize, max(r$FD$Y))) # this works
```

---

points_colorscale *Draw points with a colorscale*

Description

Draw points with values displayed as colors. Two different sets of values can be shown simultaneously: one in the background and one in the contour.

Usage

```
points_colorscale(X, Y, values,
                  bg.palette = hcl.colors(1000, "Reds 3", rev=T),
                  col.palette = hcl.colors(1000, "Reds 3",rev=T),
                  bg.range = NULL, col.range = NULL,
                  pch = 21, cex = 2, lwd = 1.5, force.range = TRUE,
                  add.col.legend = FALSE, ...)
```

Arguments

X, Y	Coordinates of the points to be displayed.
values	values of the quantity to be shown at the sampling sites. It can be a single vector (colors are shown on the inside of the points), or a data frame (the first column is related to colors on the inside, the second to colors on the outside of the points).
bg.palette, col.palette	Color palettes for values on the inside and outside of the points, respectively.
bg.range, col.range	Ranges for the legend for values on the inside and outside of the points, respectively. If not specified, the range of values (min-max) is used.
pch, cex, lwd	Same as in plot() (Note: only use values between 21-25 for pch).
force.range	Locical. If TRUE, values outside the range are constrained at the boundaries of the range; if FALSE, a transparent color is used.
add.col.legend	Logical. Add a legend for values on the outside of the points?
...	Additional arguments to be passed to imagePlot to draw the legend.

Details

A call to points is performed in the background. Therefore, a plot window must be open when this function is called.

Value

No output is produced. This function is used for its side effetcs.

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413), DEM=fp)
river <- aggregate_river(river)

# some random location of sampling sites, just to test the function
samplingSites <- c(2,15,30,78,97,117,132,106,138,153,156,159,
                   263,176,215,189,11,70,79,87,45,209,26,213)
# we use drainage area as an example variable to be shown

# 1) the function must be called after "plot(river)"
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
```

```
                    river$AG$A[samplingSites])

# 2) change color palette
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
                  river$AG$A[samplingSites],
                  bg.palette = hcl.colors(1000, "Inferno"))

# 3) impose a different range
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
                  river$AG$A[samplingSites],
                  bg.range = c(0, 1e8))

# 4) show values outside the range as transparent
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
                  river$AG$A[samplingSites],
                  bg.range = c(0, 1e8), force.range = FALSE)

# 5) show values both on inside and outside of the points (
# drainage area at the upstream vs. downstream end of the reach)
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
            data.frame(river$AG$A[samplingSites], 1.5*river$AG$A[samplingSites]),
            bg.range = c(0, 1e8), col.range = c(0, 1e8),
            lwd = 4)# increase contour line so it's more visible
# specify same range for both bg.range and col.range
# otherwise they will be shown on different scale


# 6) same as before, but show two different quantities:
# drainage area (inside) vs. elevation (outside)
# use different color palettes and add legend for the second color palette
plot(river)
points_colorscale(river$AG$X[samplingSites], river$AG$Y[samplingSites],
            data.frame(river$AG$A[samplingSites], river$AG$Z[samplingSites]),
            col.palette = terrain.colors(1000),
            lwd = 4, add.col.legend = TRUE)
```

`rast_riverweight` *Compute upstream-averaged raster*

Description

Compute a raster file where each cell value is a weighted average of upstream values.

Usage

```
rast_riverweight(x, river,
                 categorical = TRUE,
                 weightNum = list(func = "exponential",
                                  mode = "flow-based",
                                  dist50 = 500,
                                  stream = FALSE,
                                  FA = FALSE),
                 weightDen = NULL)
```

Arguments

x	SpatRaster object (obtained via terra::rast) containing categorical or continuous variables from which upstream averages are computed. Its coordinate system must be the same of river. Consider using terra::project to change coordinate system.
river	A river object generated via aggregate_river.
categorical	Logical. Is the covariate categorical (e.g. land cover classes)? If x consists of several layers, it is possible to specify categorical as a vector of logicals (one per each layer); alternatively, the single value of categorical is applied to all layers of x.
weightNum	List defining attributes of the numerator of the weighted average. See details.
weightDen	List defining attributes of the denominator of the weighted average. If NULL, it is taken equal to weightNum. See details.

Details

Lists weightNum and weightDen can include arguments func, mode, stream, FA and one between dist50, distExponential, distCauchy, distLinear, expPower. If not all of these arguments are provided, default values for weightNum are used (see examples).

func expresses the type of distance decay function used. It must be equal to one among "exponential", "cauchy", "linear", "power". Only for weightDen, the value "unweighted" is also allowed. Distance decay functions are defined as follows:

"exponential" $w(d) = \exp(1 - d/d_E)$

"cauchy" $w(d) = d_C^2/(d^2 + d_C^2)$

"linear" $w(d) = \max(1 - d/d_L, 0)$

"power" $w(d) = 1/(1 + d)^{e_P}$

"unweighted" $w(d) = 1$

where w is the weight of a given source cell, d the distance (see mode) from the source to the target cell, d_E, d_C, d_L and e_P are parameters.

mode expresses the way upstream distances are computed. It must be equal to one between "flow-based" (distances computed along steepest descent paths) and "euclidean" (i.e., distances as the crow flies).

`dist50`, `distExponential`, `distCauchy`, `distLinear`, `expPower` Parameters for the distance decay function expressed in `func`. Parameter `dist50` is the distance at which $w = 0.5$, and it can be expressed for any choice of `func`. The other parameters are specific to a given type of `func`, and are equal to the respective parameters in the formulas above (i.e., `distExponential` = d_E, `distCauchy` = d_C, `distLinear` = d_L, `expPower` = e_P). All parameters but `expPower` are distances expressed in the same unit as `x` and `river`. `expPower` is a positive, dimensionless value; note that the value of `expPower` depends on the unit of `x` and `river` (e.g., if distances in `river` are expressed in km, the same `expPower` will yield a different distance decay function than if distances in `river` are in m).

`stream` Logical. If `TRUE`, distances along the river network are not accounted for, that is, only distances (either along the steepest descent path or as the crow flies, depending on `mode`) from the source cell to the river network are considered. If `mode = "euclidean"`, this corresponds to the shortest planar distance between the source cell and any river network cell. This implies $d = 0$ for all source cells lying in the river network.

`FA` Logical. Should flow-contributing areas (expressed as numbers of cells upstream of a source cell–including the source cell itself) be included as a multiplicative factor to w?

To ensure computational efficiency for large and highly resolved rasters (note: it is the cell size of the `river` object that matters, not the resolution of `x`), it is recommended to use `func = "exponential"` (and/or `func = "exponential"` for `weightDen`) and `mode = "flow-based"`. Values of `stream` and `FA` do not affect computational speed.

Value

A `SpatRaster` object containing as many layers as the number of layers in `x`, each possibly multiplied by the number of unique categories featured in the layer (if the layer is categorical). If layer `y` `x` is categorical, the corresponding layers in the output `SpatRaster` object are named `y_z`, where `z` is the value of a unique category in the original layer `y`. If layer `y` in `x` is continuous, the corresponding layer in the output `SpatRaster` object is also named `y`.

The output `SpatRaster` object has the same extent and resolution (i.e., cell size) of the input `river` object.

See Also

`aggregate_river`, `terra::rast`, `terra::project`, `get_riverweight`

Examples

```
library(terra)
data(wigger)

r1 <- rast(system.file("extdata/landcover.tif", package = "rivnet"))
# legend: 1-urban; 2-agriculture; 3-forest; 4-improductive

r.exp <- rast_riverweight(r1, wigger)
plot(r.exp)


# unweighted denominator
r.unweighted <- rast_riverweight(r1, wigger,
```

```
weightDen = list(func = "unweighted"))

# alternative distance decay functions (with same dist50)
# these take more time than the default func = "exponential"
r.cau <- rast_riverweight(r1, wigger,
  weightNum = list(func = "cauchy"))
r.lin <- rast_riverweight(r1, wigger,
  weightNum = list(func = "linear"))
r.pow <- rast_riverweight(r1, wigger,
  weightNum = list(func = "power"))

# ignore distances on the river network
r.exp_S <- rast_riverweight(r1, wigger,
weightNum = list(stream = TRUE))

# include flow accumulation in the weight
r.exp_FA <- rast_riverweight(r1, wigger,
 weightNum = list(FA = TRUE))

# use Euclidean distances (takes more time)
# Euclidean distance from source to target
r.dO <- rast_riverweight(r1, wigger,
 weightNum = list(mode = "euclidean"))
# Euclidean distance from source to river network
r.dOS <- rast_riverweight(r1, wigger,
  weightNum = list(mode = "euclidean",
   stream = TRUE))

# specify exponential decay parameter in different ways
r.exp1 <- rast_riverweight(r1, wigger,
   weightNum = list(dist50 = 1000*log(2)))
r.exp2 <- rast_riverweight(r1, wigger,
                           weightNum = list(distExponential = 1000))
identical(r.exp1, r.exp2)
```

`river-class` *river class*

Description

A `river` object contains information on river attributes at different aggregation levels. It can represent a real river network (obtained via `extract_river`) or an optimal channel network (obtained via `OCNet::create_OCN`).

The content of a `river` object can be treated as a list, hence its objects and sublists can be accessed with both the `$` and `@` operators.

For information on the aggregation levels and on the content of a `river` object, see `OCNet::OCNet-package`.

Examples

```
fp <- system.file("extdata/wigger.tif", package="rivnet")
r <- extract_river(outlet=c(637478,237413),
DEM=fp)

show(r)
names(r)

# extract or replace parts of a river object
r$dimX
r@dimX
dim <- r[["dimX"]]
r$dimX <- 1
r[["dimX"]]
r[["dimX"]] <- dim
```

`river_to_AEM` *river_to_AEM*

Description

Construct asymmetric eigenvector maps (AEM) from a river

Usage

```
river_to_AEM(river, ...)
```

Arguments

`river`	A `river` object.
`...`	Further arguments to be passed to `OCN_to_AEM`.

Details

This is an alias to `OCNet::OCN_to_AEM`.

Value

A `river` object.

Examples

```
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478, 237413),
DEM = fp)
r <- aggregate_river(r)
out.aem <- river_to_AEM(r)
```

`river_to_igraph` *river_to_igraph*

Description

Transform a `river` in an `igraph` object.

Usage

```
river_to_igraph(river, ...)
```

Arguments

`river`	A `river` object.
`...`	Further arguments to be passed to `OCN_to_igraph`.

Details

This is an alias to `OCNet::OCN_to_igraph`.

Value

An `igraph` object.

Examples

```
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478,237413),
DEM=fp)
r <- aggregate_river(r)
g <- river_to_igraph(r, level = "AG")
g
```

`river_to_shapefile` *Export river network as shapefile*

Description

Export river network as shapefile. Reach attributes can be added.

Usage

```
river_to_shapefile(river, filename, atts = NULL,
                   EPSG = NULL, ...)
```

Arguments

`river`	`river` object. It must have been aggregated (via `aggregate_river`).
`filename`	Character. Output file name. It must contain the ".shp" extension.
`atts`	Attributes at AG level that can be exported. This should be a character vector, with entries corresponding to the names of fields within the `AG` sub-list of `river`. See example.
`EPSG`	EPSG code. Default is `NULL`, which does not produce a .prj file (i.e., the shapefile does not contain projection information).
`...`	Additional arguments to be passed to `writeVector` (e.g., `overwrite = TRUE` allows overwriting an existing shapefile).

Value

No output is produced. This function is used for its side effetcs.

See Also

`terra::writeVector`

Examples

```
library(terra) # to use "vect"
fp <- system.file("extdata/wigger.tif", package="rivnet")
river <- extract_river(outlet=c(637478,237413), DEM=fp)
river <- aggregate_river(river)

tmpname <- paste0(tempfile(), ".shp")
river_to_shapefile(river, tmpname, overwrite = TRUE)

# read output
vv <- vect(tmpname)
vv
plot(vv)
```

```
# add attributes to shapefile (drainage area, stream order)
river_to_shapefile(river, tmpname,
atts = c("A", "streamOrder"), # same names as in river$AG
overwrite = TRUE)
vv <- vect(tmpname)
vv

# add projection
river_to_shapefile(river, tmpname,
EPSG = 21781,
overwrite = TRUE)
vv <- vect(tmpname)
vv
```

`river_to_SSN` *Transform river into SSN object*

Description

Transform a `river` in a `SpatialStreamNetwork` object.

Usage

```
river_to_SSN(river, ...)
```

Arguments

`river`	A `river` object.
`...`	Further arguments to be passed to `OCN_to_SSN`.

Details

This is an alias to `OCNet::OCN_to_SSN`.

Value

A `SpatialStreamNetwork` object if `importToR` is `TRUE`, otherwise `NULL`.

Examples

```
 fp <- system.file("extdata/wigger.tif", package="rivnet")
 r <- extract_river(outlet=c(637478,237413),
DEM=fp)
r <- aggregate_river(r)
```

```
s <- river_to_SSN(r, level = "AG", obsSites = sample(r$AG$nNodes, 10),
path = paste(tempdir(),"/river.ssn", sep = ""), importToR = TRUE)
plot(s)
```

---

wigger *River Wigger*

Description

It is built via

```
wigger <- extract_river(outlet=c(637478,237413), EPSG=21781, ext=c(6.2e5,6.6e5,2e5,2.5e5),
z=9)
wigger <- aggregate_river(wigger, maxReachLength = 2500)
hydrodata <- data.frame(data=c(8, 15), type=c("w","Q"), node=wigger$AG$outlet*c(1,1))
wigger <- hydro_river(hydrodata, wigger)
r1 <- rast(system.file("extdata/landcover.tif", package="rivnet"))
wigger <- covariate_river(r1, wigger)
```

Usage

```
data(wigger)
```

Format

A `river` object. See `extract_river` documentation for details.

Index