Package ‘openSTARS’

May 21, 2020

Type Package

Title An Open Source Implementation of the 'ArcGIS' Toolbox 'STARS'

Description An open source implementation of the 'STARS' toolbox
(Peterson & Ver Hoef, 2014, <doi:10.18637/jss.v056.i02>) using 'R' and 'GRASS GIS'.
It prepares the *.ssn object needed for the 'SSN' package.
A Digital Elevation Model (DEM) is used to derive stream networks
(in contrast to 'STARS' that can clean an existing stream network).

Version 1.2.0

URL `https://github.com/MiKatt/openSTARS`

Depends R (>= 3.3), data.table, rgrass7

License MIT + file LICENSE

Encoding UTF-8

LazyLoad true

LazyData true

Imports methods, progress, rgdal, sp, raster, SSN

RoxygenNote 7.1.0

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Mira Kattwinkel [aut, cre],
Eduard Szöcs [aut]

Maintainer Mira Kattwinkel <`mira.kattwinkel@gmx.net`>

Repository CRAN

Date/Publication 2020-05-21 19:30:03 UTC

R topics documented:

`calc_attributes_edges` *Calculate attributes of the edges.*

Description

For each edge (i.e. stream segment) additional attributes (potential predictor variables) are derived based on given raster or vector maps.

Usage

```
calc_attributes_edges(
  input_raster = NULL,
  stat_rast = NULL,
  attr_name_rast = NULL,
  input_vector = NULL,
  stat_vect = NULL,
  attr_name_vect = NULL,
  round_dig = 2
)
```

Arguments

`input_raster`	name(s) of raster map(s) to calculate attributes from.
`stat_rast`	name(s) giving the statistics to be calculated, from the raster maps, must be one of: "min", "max", "mean", "sum", "percent", "area" for each `input_raster`.
`attr_name_rast`	of new column name(s) for the attribute(s) to be calculated. Attribute names must not be longer than 8 characters as ESRI shapefiles cannot have colum names with more than 10 characters. See notes.
`input_vector`	name(s) of vector map(s) to calculate attributes from.
`stat_vect`	name(s) giving the statistics to be calculated from the vector maps, must be one of: "count" (for point data), "percent" or "area" (for polygon data) for each `input_vector`.
`attr_name_vect`	name(s) of attribute column(s), case sensitive. For polygon data, this is the column to calculate the statistics from; the results column names are created by the content of this column. For point data, a column will be created with this name to hold the counts. See notes.
`round_dig`	integer; number of digits to round results to. Can be a vector of different values or just one value for all attributes. #@param clean logical; should intermediate files be deleted

Details

First, the reach contributing areas (= subcatchments) for all edges are calculated. Then these are intersected with the given raster and/or vector maps and the desired statistics are computed. This function must be run before computing approximate attribute values for sites `calc_attributes_sites_approx`.

For `stat_rast` = "percent" or "area" the `input_raster` can be either coded as 1 and 0 (e.g., cells occupied by the land use under consideration and not) or as different classes. The percentage or area of each class in the catchment is calculated. If the `input_raster` consists of percentages per cell (e.g., proportional land use of a certain type per cell) `stat_rast` = "mean" gives the overall proportion of this land use in the catchment.

For `stat_vect` = "percent" or "area" `input_vector` must contain polygons of e.g. different land use types. The column `attr_name_vect` would then give the code for the different land uses. Then, the percentage for each land use type in the catchment of the edge is calculated and given in separate columns with column names resampling the different categories given in column `attr_name_vect`

For `stat_vect` = "count" `input_vector` must contain points of e.g. waste water treatment plants. The column `attr_name_vect` gives the name of the column to hold the count value, e.g. nWWTP.

Both raster and vector maps to be used must be read in to the GRASS session, either in `import_data` or using the GRASS function r.in.rast or v.in.ogr (see examples).

Value

Nothing. The function appends new columns to the 'edges' attribute table with column names given in `attr_name_rast` and derived from the attribute classes for vector data. For each attribute, two columns are appended: one giving the attribute for the rca of the edge ("attribute_name_e") and one for the attribute of the total catchment of the edge ("attribute_name_c").

Note

Column names for the results are created as follows: Raster data - the column names given in `attr_name_rast` are used. The user should take care to use unique, clear names. For `stat_rast` = 'percentage' or 'area', the output column name will be concatenated 'p' or 'a', repectively. For vector data, column names are constructed from the entries in in the column `attr_name_vect`. For counts of points, the new column name containing the counts is just the given name. For polygon data ('percentage' or 'area'), the names are constructed using the unique entries of the column with a concatenated 'p' or 'a', repectively. If, for instance, for a landuse vector containing the classes 'urban' and 'arable' percentages would be calculated, edges would contain two new columns 'urbanp' and 'arablep'.

`setup_grass_environment`, `import_data`, `derive_streams` and `calc_edges` must be run before.

Author(s)

Mira Kattwinkel, <`mira.kattwinkel@gmx.net`>

Examples

```
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
streams_path <- system.file("extdata", "nc", "streams.shp", package = "openSTARS")
preds_v_path <- system.file("extdata", "nc", "pointsources.shp", package = "openSTARS")
preds_r_path <- system.file("extdata", "nc", "landuse_r.tif", package = "openSTARS")


setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path, streams = streams_path,
            predictor_vector = preds_v_path, predictor_raster = preds_r_path)
gmeta()

# Derive streams from DEM
# burn in 'streams' 10 meters
derive_streams(burn = 10, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# calculate slope as potential predictor
execGRASS("r.slope.aspect", flags = c("overwrite","quiet"),
parameters = list(
  elevation = "dem",
    slope = "slope"
    ))


# Prepare edges
calc_edges()
calc_attributes_edges(input_raster = c("slope", "landuse_r"),
                      stat_rast = c("max", "percent"),
                      attr_name_rast = c("maxSlo", "luse"),
                      input_vector = "pointsources", stat_vect = "count",
                      attr_name_vect = "psource")

# Plot eges with percentage of forest in the catchment (lusep_5) as line width
edges <- readVECT('edges', ignore.stderr = TRUE)
head(edges@data)
lu <- readRAST("landuse_r", ignore.stderr = TRUE)

 # plot landuse data
library(raster)
op <- par()
par(xpd = FALSE)
plot(raster(lu), legend = FALSE, xaxt = "n", yaxt = "n", bty = "n",
col = adjustcolor(c("red", "goldenrod", "green", "forestgreen",
"darkgreen", "blue", "lightblue"), alpha.f = 0.7))
par(xpd = TRUE)
legend("bottom", cex = 0.75,
 legend = c("developed", "agriculture", "herbaceous", "shrubland", "forest", "water", "sediment"),
  fill = c("red", "goldenrod", "green", "forestgreen","darkgreen", "blue", "lightblue"),
  horiz = TRUE, inset = -0.175)
plot(edges, lwd = edges$lusep_5_c * 10, add = TRUE)
par <- op
```

calc_attributes_sites_approx

Calculate attributes of the sites.

Description

For each site (observations or predictions) attributes (potential predictor variables) are derived based on the values calculated for the edge the site lies on. This function calculates approximate values for site catchments as described in Peterson & Ver Hoef, 2014: STARS: An ArcGIS Toolset Used to Calculate the Spatial Information Needed to Fit Spatial Statistical Models to Stream Network Data. J. Stat. Softw., 56 (2).

Usage

```
calc_attributes_sites_approx(
  sites_map = "sites",
  input_attr_name = NULL,
  output_attr_name = NULL,
  stat = NULL,
  round_dig = 2,
  calc_basin_area = TRUE
)
```

Arguments

`sites_map`	character; name of the sites (observation or prediction) attributes shall be calculated for. "sites" (default) refers to the observation sites.
`input_attr_name`	character vector; input column name(s) in the attribute table of the vector map "edges".
`output_attr_name`	character vector (optional); output column name(s) appended to the site attribute data table. If not provided it is set to `input_attr_name`. Attribute names must not be longer than 10 characters.
`stat`	name or character vector giving the statistics to be calculated. See details below.
`round_dig`	integer; number of digits to round results to.
`calc_basin_area`	boolean; shall the catchment area be calculated? (Useful to set to FALSE if the function has been called before.)

Details

The approximate total catchment area (H2OAreaA) is always calculated if `calc_basin_area` is TRUE. If `stat` is one of "min", "max", "mean" or "percent" the function assigns the value of the edge the site lies on. Otherwise, the value is calculated as the sum of all edges upstream of the previous confluence and the proportional value of the edge the site lies on (based on the distance ratio 'ratio'); this is useful e.g. for counts of dams or waste water treatment plants or total catchment area.

`input_attr_name` must give the column names of the edges attribute table for that the statistics should be calculated.

Value

Nothing. The function appends new columns to the `sites_map` attribute table

- 'H2OAreaA': Total watershed area of the watershed upstream of each site.
- attr_name: Additional optional attributes calculated based on `input_attr_name`.

Note

`import_data`, `derive_streams`, `calc_edges`, `calc_sites` or `calc_prediction_sites` and `calc_attributes_edges` must be run before.

Author(s)

Mira Kattwinkel, <mira.kattwinkel@gmx.net>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
pred_path <- system.file("extdata", "nc", "geology.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path,
 predictor_vector = pred_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Derive slope from the DEM as an example raster map to calculate attributes from
execGRASS("r.slope.aspect", flags = c("overwrite","quiet"),
parameters = list(
  elevation = "dem",
    slope = "slope"
    ))
calc_attributes_edges(input_raster = "slope", stat_rast = "max", attr_name_rast = "maxSlo",
              input_vector = "geology", stat_vect = "percent", attr_name_vect = "GEO_NAME")

calc_sites()

# approximate potential predictor variables for each site based on edge values
calc_attributes_sites_approx(
input_attr_name = c('maxSlo', 'CZamp', 'CZbgp', 'CZfgp', 'CZgp', 'CZigp', 'CZlgp', 'CZvep', 'Kmp'),
  stat = c("max", rep("percent", 8)))

# plot share of a certain geology in the sampling point's catchment as
# point size
library(sp)
edges <- readVECT('edges', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
geo <- readVECT("geology", ignore.stderr = TRUE)
plot(geo, col = adjustcolor(1:8, alpha.f = 0.5)[as.factor(geo$GEO_NAME)])
plot(edges, col = "blue", add = TRUE)
plot(sites, col = 1, add = TRUE, pch = 19, cex = (sites$CZbgp + 0.15) * 2)
legend("left", col = adjustcolor(1:8, alpha.f = 0.5), bty = "n",
legend = unique(geo$GEO_NAME), pch = 15, title = "geology")
legend("right", col = 1, pch = 19, legend = seq(0, 1, 0.2), bty = "n",
title = "share CZbg\nin catchment", pt.cex =  (seq(0, 1, 0.2) + 0.15) * 2)
```

calc_attributes_sites_exact

Calculate attributes of the sites.

Description

For each site (observation or prediction) the total catchment area is calculated ('H2OArea'). Additionally, other attributes (predictor variables) can be derived based on given raster or vector maps. This function calculates exact values for catchments derived with r.stream.basins and can take considerable time if there are many sites. Catchment raster maps can optionally be stored as "sitename_catchm_X" (X = locID).

Usage

```
calc_attributes_sites_exact(
  sites_map = "sites",
  input_raster = NULL,
  stat_rast = NULL,
  attr_name_rast = NULL,
  input_vector = NULL,
  stat_vect = NULL,
  attr_name_vect = NULL,
  round_dig = 2,
  calc_basin_area = TRUE,
  keep_basins = FALSE
)
```

Arguments

`sites_map`	character; name of the sites (observation or prediction) attributes shall be calculated for. "sites" (default) refers to the observation sites.

`input_raster`	character vector (optional); name of additional raster maps to calculate attributes from.
`stat_rast`	character vector (optional); statistics to be calculated, one of: "min", "max", "mean", "stddev", "variance", "sum", "median", "percent", "area" or "percentile_X" (where X gives the desired percentile e.g. 25 for the first). Must be provided if `input_raster` are given.
`attr_name_rast`	character vector (optional); column name for the attributes to be calculated. Attribute names must not be longer than 10 characters. Must be provided if `input_raster` are given.
`input_vector`	character string vector (optional); name of additional vector maps to calculate attributes from.
`stat_vect`	character string vector (optional); statistics to be calculated, one of: "percent", "area" or "count." Must be provided if `input_vector` is given.
`attr_name_vect`	character string vector (optional); column name(s) in the vector file provided to calculate the attributes from (if `input_vector` is a polygon map and stat_vect is "percent") or giving the new name attributes to calculate (if `input_vector` is a point map and stat_vect is "count". Must be provided if `input_vector` is given.
`round_dig`	integer; number of digits to round results to. Can be a vector of different values or just one value for all attributes.
`calc_basin_area`	boolean; shall the catchment area be calculated? (Useful to set to FALSE if the function has been called before with `keep_basins = TRUE`.)
`keep_basins`	boolean; shall raster and vector maps of all the watersheds be kept? Defaults to FALSE.

Value

Nothing. The function appends new columns to the `sites_map` attribute table

- 'H2OArea': Total watershed area of the watershed upstream of each site.
- attr_name_rast: Additional optional attributes calculated based on input_raster maps.
- attributes form vector maps:Additional optional attributes calculated based on input_vector maps. The column names are based on the unique entries of the column(s) given in `attr_name_vect`.

Please note that for sampling points that lie in the same DEM raster cell along a stream identical values are calculated because identical watersheds are derived.

Note

`import_data`, `derive_streams`, `calc_edges` and `calc_sites` or `calc_prediction_sites` must be run before.

If `calc_basin_area = F` but there are no raster maps called 'sitename_catchm_x' with x = locID of all sites the catchments (and their area) are derived.

Author(s)

Mira Kattwinkel, `<mira.kattwinkel@gmx.net>`, Eduard Szoecs, `<eduardszoecs@gmail.com>`

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
} else {
  gisbase = "/usr/lib/grass74/"
}
initGRASS(gisBase = gisbase,
     home = tempdir(),
     override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Prepare edges
calc_edges()
execGRASS("r.slope.aspect", flags = c("overwrite","quiet"),
          parameters = list(
            elevation = "dem",
            slope = "slope"
          ))
calc_attributes_edges(input_raster = "slope", stat_rast = "max", attr_name_rast = "maxSlo")

# Prepare sites
calc_sites()
calc_attributes_sites_approx(input_attr_name = "maxSlo", output_attr_name = "maxSloA", stat = "max")
calc_attributes_sites_exact(input_raster = "slope", attr_name_rast = "maxSloE", stat_rast = "max")

# Plot data
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
edges <- readVECT('edges', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
plot(dem, col = gray(seq(0,1,length.out=20)))
mm <- range(c(edges$maxSlo_e, sites$maxSloA, sites$maxSloE))
b <- seq(from = mm[1], to = mm[2] + diff(mm) * 0.01, length.out = 10)
c_ramp <- colorRampPalette(c("white", "blue", "orange", "red"))
cols <- c_ramp(length(b))[as.numeric(cut(edges$maxSlo_e, breaks = b, right = FALSE))]
# plot stream edges, color depending on maxSlope of the edge
plot(edges,col = cols, lwd = 2, add = TRUE)
cols <- c_ramp(length(b))[as.numeric(cut(sites$maxSloA,breaks = b,right = FALSE))]
# plot sites as points with color corresponding to maxSlop approximate
plot(sites, pch = 19, col = cols, cex = 2, add = TRUE)
cols <- c_ramp(length(b))[as.numeric(cut(sites$maxSloE,breaks = b,right = FALSE))]
```

```
#' # plot sites as ring around points with color corresponding to maxSlop exact
plot(sites, pch = 21, bg = cols, cex = 1.1, add = TRUE)
# Some points in the lower centre of the map indicate a difference in max slope between
# approximate and exact calculation (different colors for inner and outer points). However,
# for most points the values are similar
```

`calc_binary` *Calculate binary IDs for each stream network.*

Description

Calculate binary IDs for each stream network built up by '0' and '1'. This function is called by `export_ssn` and there is no need for it be called by the users.

Calculate binary IDs for each stream network built up by '0' and '1'. This function is called by `export_ssn` and there is no need for it be called by the users.

Usage

```
calc_binary()
```

```
calc_binary()
```

Value

A list with one slot for each network id containing a data frame with 'rid' and 'binaryID' for each segment belonging to this network.

A list with one slot for each network id containing a data frame with 'rid' and 'binaryID' for each segment belonging to this network.

Note

`import_data`, `derive_streams`, `calc_edges` and `calc_sites` must be run before.

`import_data`, `derive_streams`, `calc_edges` and `calc_sites` must be run before.

Author(s)

Eduard Szoecs, <`eduardszoecs@gmail.com`>; Mira Kattwinkel, <`mira.kattwinkel@gmx.net`>

Eduard Szoecs, <`eduardszoecs@gmail.com`>; Mira Kattwinkel, <`mira.kattwinkel@gmx.net`>

@export

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Prepare site
calc_sites()

binaries <- calc_binary()
head(binaries[[1]])
```

calc_catchment_attributes_rast

calc_catchment_attributes_rast Aggregate attributes for the total catchment of each stream segment.

Description

This function aggregates the attributes of each segment for the total catchment of each stream segment. It is called within `calc_attributes_edges` and should not be called by the user.

Usage

```
calc_catchment_attributes_rast(dt, stat_rast, attr_name_rast, round_dig)
```

Arguments

`dt`	data.table of stream topology and attributes per segment.
`stat_rast`	name or character vector giving the statistics to be calculated, must be one of: min, max, mean, percent, sum.
`attr_name_rast`	name or character vector of column names for the attribute(s) to be calculated.
`round_dig`	integer; number of digits to round results to. Can be a vector of different values or just one value for all attributes.

Value

Nothing. The function changes the values of the columns attr_name_rast in dt.

`calc_catchment_attributes_rast_rec`

calc_catchment_attributes_rast_rec Aggregate attributes for the total catchment of each stream segment.

Description

Recursive function to calculate the catchment attributes of each stream segment. It is called by `calc_catchment_attributes_rast` for each outlet and should not be called by the user.

Usage

```
calc_catchment_attributes_rast_rec(dt, id, stat, attr_name)
```

Arguments

`dt`	data.table of stream topology and attributes per segment.
`id`	integer; 'stream' of outlet segment to start the calculation from.
`stat`	name or character vector giving the statistics to be calculated, must be one of: min, max, mean, percent.
`attr_name`	name or character vector of column names for the attribute(s) to be calculated.

Value

One row data.table with the cumulative number of cells of the total catchment of each segment and the values for each attribute and changes the values in dt.

Note

The values for stats "mean" and "percent" need to be divided by the cumulative number of cells of the total catchment in a subsequent step.

`calc_catchment_attributes_vect` *calc_catchment_attributes_vect Aggregate attributes for the total catchment of each stream segment.*

Description

This function aggregates the attributes of each segment for the total catchment of each stream segment. It is called within `calc_attributes_edges` and should not be called by the user.

Usage

```
calc_catchment_attributes_vect(dt, stat_vect, attr_name_vect, round_dig)
```

Arguments

`dt`	data.table of stream topology and attributes per segment.
`stat_vect`	name or character vector giving the statistics to be calculated, must be one of: percent, sum.
`attr_name_vect`	name or character vector of column names for the attribute(s) to be calculated.
`round_dig`	integer; number of digits to round results to. Can be a vector of different values or just one value for all attributes.

Value

Nothing. The function changes the values of the columns attr_name_vect in dt.

`calc_edges` *Calculate edges for SSN object.*

Description

A vector (lines) map ’edges’ is derived from ’streams_v’ and several attributes are assigned.

Usage

```
calc_edges()
```

Details

Steps include:

- Assign unique ’rid’ to each stream segment
- Find different stream networks in the region and assign ’netID’
- Calculate segments upstream distance, ’upDist’ = flow length from the upstream node of the stream segment to the outlet of the network
- Calculate reach contributing areas (RCA) per segment, ’rcaArea’ = subcatchment area of each segment in square km
- Calculate catchment areas, ’H2OArea’ = total catchment area of each segment in square km

All lengths are rounded to 2 and all areas to 6 decimal places, respectively.

Value

Nothing. The function produces the following map:

- ’edges’: derived stream segments with computed attributes needed for ’SSN’ (vector)

Note

`setup_grass_environment`, `import_data` and `derive_streams` must be run before.

Author(s)

Mira Kattwinkel, `<mira.kattwinkel@gmx.net>`, Eduard Szoecs, `<eduardszoecs@gmail.com>`

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)
```

```
check_compl_confluences()

# Prepare edges
calc_edges()

# Plot data
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
edges <- readVECT('edges', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20))
lines(edges, col = 'blue')
```

`calc_prediction_sites` *Calculate prediction sites for 'SSN' object.*

Description

A vector (points) map of prediction sites is created and several attributes are assigned.

Usage

```
calc_prediction_sites(predictions, dist = NULL, nsites = 10, netIDs = NULL)
```

Arguments

`predictions`	string giving the name for the prediction sites map.
`dist`	number giving the distance between the points to create in map units.
`nsites`	integer giving the approximate number of sites to create
`netIDs`	integer (optional): create prediction sites only on streams with these netID(s).

Details

Either `dist` or `nsites` must be provided. If `dist` is NULL, it is estimated by dividing the total stream length in the map by `nsites`; the number of sites actually derived might therefore be a bit smaller than `nsites`.

Steps include:

- Place points on edges with given distance from each other
- Save the point coordinates in NEAR_X and NEAR_Y.
- Assign unique identifiers (needed by the 'SSN' package) 'pid' and 'locID'.
- Get 'rid' and 'netID' of the stream segment the site intersects with (from map 'edges').
- Calculate upstream distance for each point ('upDist').
- Calculate distance ratio ('distRatio') between position of the site on the edge (= distance traveled from lower end of the edge to the site) and the total length of the edge.

'pid' and 'locID' are identical, unique numbers. 'upDist' is calculated using r.stream.distance. Points are created using v.segment.

Note

import_data, derive_streams and calc_edges must be run before.

Author(s)

Mira Kattwinkel <mira.kattwinkel@gmx.net>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

check_compl_confluences()
calc_edges()
calc_sites()
calc_prediction_sites(predictions = "preds", dist = 2500)

library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
preds <- readVECT('preds', ignore.stderr = TRUE)
edges <- readVECT('edges', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20))
lines(edges, col = 'blue', lwd = 2)
points(sites, pch = 4)
points(preds, pch = 19, col = "steelblue")
```

---

calc_sites — *Calculate sites for SSN object.*

Description

A vector (points) map 'sites' is derived and several attributes are assigned.

Usage

```
calc_sites(locid_col = NULL, pid_col = NULL, pred_sites = NULL, maxdist = NULL)
```

Arguments

`locid_col`	character (optional); column name in the sites attribute table giving a unique site identifier. If not provided, it is created automatically (based on the 'cat' field; default).
`pid_col`	character (optional); column name in the sites attribute table that distinguishes between repeated measurements at a sampling site, e.g. by date. If not provided, it is created automatically.
`pred_sites`	character vector (optional); names for prediction sites (loaded with `import_data`).
`maxdist`	integer (optional); maximum snapping distance in map units (see details). Sites farther away from edges will be deleted.

Details

Steps include:

- Snap points to derived network (edges). 'dist' gives the distance of the original position to the closest streams segment. If this is a too large value consider running `derive_streams` again with smaller value for `accum_threshold` and/or `min_stream_length`.
- Save the new point coordinates in NEAR_X and NEAR_Y.
- Assign unique 'pid' and 'locID' (needed by the 'SSN' package).
- Get 'rid' and 'netID' of the stream segment the site intersects with (from map "edges").
- Calculate upstream distance for each point ('upDist').
- Calculate distance ratio ('ratio') between position of site on edge (distance traveled from lower end of the edge to the site) and the total length of the edge.

Often, survey sites do not lay exactly on the stream network (due to GPS imprecision, stream representation as lines, derivation of streams from dem, etc.). To assign an exact position of the sites on the network they are moved to the closest stream segment (snapped) using the GRASS function v.distance.

If `locid_col` and `pid_col` are not provided, 'pid' and 'locID' are identical, unique numbers. If they are provided, they are created based on these columns (as numbers, not as text). Note that measurements can be joined to the sites at a later step (including multiple parameters and multiple measurements) using `merge_sites_measurements`. Then, 'pid' is updated accordingly.

'upDist' is calculated using v.distance with upload = "to_along" which gives the distance along the stream segment to the next upstream node ('distalong'). 'upDist' is the difference between the 'upDist' of the edge the point lies on and 'distalong'.

The unit for distances (= map units) can be found out using execGRASS("g.proj", flags = "p").

If prediction sites have been created outside of this package they can be processed here as well. They must have been imported with `import_data` before. Alternatively, prediction sites can be created using `calc_prediction_sites`.

Note

`import_data`, `derive_streams` and `calc_edges` must be run before.

Author(s)

Mira Kattwinkel <`mira.kattwinkel@gmx.net`>, Eduard Szoecs, <`eduardszoecs@gmail.com`>,

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Prepare site
calc_sites()

# Plot data
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
```

```
edges <- readVECT('edges', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
sites_o <- readVECT('sites_o', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20),axes = TRUE)
lines(edges, col = 'blue')
points(sites, pch = 4)
points(sites_o, pch = 1)
legend("topright", pch = c(1, 4), legend = c("original", "corrected"))
```

`check_compl_confluences` *Check if there are more than two inflows to an outflow.*

Description

It is checked, if more than two line segments flow into a node, i.e. if there are more than two inflows to an outflow.

Usage

```
check_compl_confluences()
```

Details

It is checked, if there are columns named 'prev_str03', 'prev_str04' and 'prev_str05' in the attribute table of streams_v derived with `derive_streams` (i.e. based on the GRASS function r.stream.order).

Value

TRUE if there are complex confluences.

Note

`setup_grass_environment`, `import_data` and `derive_streams` must be run before.

Author(s)

Mira Kattwinkel <`mira.kattwinkel@gmx.net`>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
```

```
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

check_compl_confluences()
```

check_projection *Compare projection raster data to the one of the current GRASS location.*

Description

Check if the procection of raster files matches the one of the current region

Usage

```
check_projection(path)
```

Arguments

`path`	character string vector; path raster data file(s)

Details

Prints out a table of the PROJ.4 elements of the projection information of the current GRASS location and of the raster file(s) as well as one columns for each comparison indicating the differences. Based on this information it can be decided if the data can be read into GRASS (`import_data`) without prior processing, i.e. if all raster data are of the same projection.

Value

Nothing.

check_ssn *Checking 'SSN' object.*

Description

This function roughly checks the 'SSN' object. It returns FALSE if some essential columns are missing or values have illegal values.

Usage

```
check_ssn(path, predictions = NULL)
```

Arguments

`path`	character; path to .ssn object.
`predictions`	name(s) of prediction map(s) (optional).

Value

TRUE or FALSE depending if checks pass.

Author(s)

Mira Kattwinkel, <mira.kattwinkel@gmx.net>, Eduard Szoecs, <eduardszoecs@gmail.com>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Prepare site
calc_sites()
# Calculate H2OArea
calc_attributes_sites_exact()

# Plot data
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
sites_orig <-  readVECT('sites_o', ignore.stderr = TRUE)
edges <- readVECT('edges', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20))
lines(edges, col = 'blue')
points(sites_orig, pch = 4)
cols <- colorRampPalette(c("blue", 'red'))(length(sites$H2OArea))[rank(sites$H2OArea)]
points(sites, pch = 16, col = cols)

# Write data to SSN Folder
ssn_dir <- file.path(tempdir(), 'nc.ssn')
export_ssn(ssn_dir, delete_directory = TRUE)

# Check if all files are ok
library(SSN)
check_ssn(ssn_dir)
```

`correct_compl_confluences`

Correct confluences with three or more inflows.

Description

At complex confluences (when more than two line segments flow into a node, i.e. more than two inflows to an outflow), the end of one of the inflows is moved a tiny bit upstream to one of the other inflows to create a new confluence of two streams (see details below).

Usage

```
correct_compl_confluences(clean = TRUE)
```

Arguments

clean	logical; should intermediate files be removed from 'GRASS' session?

Details

At complex confluences (when more than two line segments flow into a node, i.e. more than two inflows to an outflow), new confluences of only two streams are created: 1. complex confluences are found based on the fact that the outflow has more than two previous streams 2. the inflow with the shortest cummulative length from its source is found; the end of this segment will be moved 3. the inflow with the smallest angle to this inflow is found; this segment will be cut into tow segments close to the juntion using the GRASS function v.edit(tool = break) creating a new confluence 4. the shortest inflow found in 2 is moved to the newly created confluence using v.edit(tool = vertexmove) 5. all lengths are updated (segment length, cumulative length, i.e. length of the stream from the source, distance to the outlet). The distance the shortest confluence is moved depends on the number of inflows. For three inflows, it is moved 1/12 time the DEM cellsize upstream, for seven (the extremly rare maximum) 5/12 * cellsize.

Value

Nothing. The function changes features in 'streams_v'. Changed features are marked in the new column 'changed'.

Note

`setup_grass_environment`, `import_data` and `derive_streams` must be run before.

Author(s)

Mira Kattwinkel <`mira.kattwinkel@gmx.net`>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
streams_path <- system.file("extdata", "nc", "streams.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path, streams = streams_path)
gmeta()
```

```
# Derive streams from DEM
derive_streams(burn = 10, accum_threshold = 100, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are complex confluences in the
# example date set if the accumulation threshold is low)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# plot
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
streams <- readVECT('streams_v', ignore.stderr = TRUE)
streams_orig <- readVECT('streams_v_o3', ignore.stderr = TRUE)
# zoom to a relevant part of the dem
plot(dem, col = terrain.colors(20), axes = TRUE,
  xlim = c(640100,640150), ylim = c(219735,219785))
lines(streams_orig, col = 'red', lwd = 4)
lines(streams, col = 'blue', lty = 2, lwd = 2)
legend("bottomright", col = c("red", "blue"), lty = c(1,2), lwd = c(4,2),
  legend = c("original", "corrected"))

plot(streams, col = c("blue", "red")[streams@data$changed+1], lty = 1, lwd = 2)
```

`delete_lakes` *Delete lakes from stream network*

Description

When the stream network is derived from a dem, the streams will just cross lakes or ponds. However, the flow is disconnected here and the relationship between sampling points upstream and downstream of a lake is not clear. For instance, chemicals could be retained and temperature altered in a lake. This function intersects the stream network with a given vector map of lakes; it deletes the stream segments in the lake, breaks those that cross its borders and assigns a new, updated topology.

Usage

```
delete_lakes(lakes, keep = TRUE)
```

Arguments

`lakes`	character string or object; path to lake vector file (ESRI shape), name of vector map in the GRASS data base or sp or sf data object.
`keep`	boolean; should the original 'streams_v' be saved? Default is TRUE.

Value

Nothing. The function updates 'streams_v' and (if keep = TRUE) saves the original file to streams_v_prev_lakes. If `lakes` is a file path, the lakes are imported into GRASS as 'lakes'.

Note

The column 'out_dist' (flow length from the upstream node of the segment to the outlet of the network) is updated based on the new segment length. In contrast, 'cum_length' is not updated as it is no longer needed and will be deleted in calc_edges.

#'

Author(s)

Mira Kattwinkel <`mira.kattwinkel@gmx.net`>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
lakes_path <- system.file("extdata", "nc", "lakes.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 100, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

delete_lakes(lakes = lakes_path)

# plot
library(sp)
```

```
dem <- readRAST('dem', ignore.stderr = TRUE)
streams <- readVECT('streams_v', ignore.stderr = TRUE)
streams_with_lakes <- readVECT('streams_v_prev_lakes', ignore.stderr = TRUE)
lakes <- readVECT('lakes', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20), axes = TRUE)
lines(streams_with_lakes, col = 'red', lty = 2, lwd = 2)
lines(streams, col = 'blue', lty = 4, lwd = 2)
legend("topright", col = c("red", "blue"), lty = c(1,4), lwd = c(2,2),
  legend = c("through lakes", "lakes cut out"))
```

`derive_streams` *Derive stream network from DEM.*

Description

Streams are derived from a digital elevation model (DEM) using the GRASS function r.stream.extract. If a stream network is available (see `import_data`) and burn > 0 it will be first burnt into DEM. Stream topology is derived using the GRASS function r.stream.order.

Usage

```
derive_streams(
  burn = 0,
  accum_threshold = 700,
  condition = TRUE,
  min_stream_length = 0,
  dem_name = NULL,
  clean = TRUE,
  mem = FALSE
)
```

Arguments

`burn`	numeric; how many meters should the streams be burned into the DEM? Only applicable if a mapped stream network is provided in `import_data`. Defaults to 0.
`accum_threshold`	integer; accumulation threshold to use (i.e. minimum flow accumulation value in cells that will initiate a new stream). A small value results in many small streams. Defaults to 700 but a reasonable value strongly depends on the raster resolution. See details below.
`condition`	logical; should the DEM be conditioned using the GRASS function r.hydrodem; default: TRUE.
`min_stream_length`	integer: minimum stream length in number of DEM raster cells; shorter first order stream segments are deleted. Defaults to 0 but a reasonable value strongly depends on the raster resolution. See details below.

`dem_name`	character vector, optional; default: 'dem'; useful if conditioned and / or burnt in DEM raster from previous runs shall be used.
`clean`	logical; should intermediate raster layer of imported streams ('streams_or') be removed from the GRASS session? Defaults to TRUE.
`mem`	logical; should -m flag in the GRASS function r.watershed be used (for data preparation)? Defaults to FALSE; if set to TRUE the calculation uses disk swap mode, i.e. it is not carried out in the RAM but also using disk space. Useful for large data sets but also slower.

Details

For details on `accum_threshold` and `min_stream_length` see the parameters 'threshold' and 'stream_length' at r.stream.extract. It might be useful to not burn in the whole available stream network but only parts of it (e.g., larger streams with higher Strahler stream order only). For this, the stream network needs to be pre-processed (parts could be deleted) before loading it with `import_data`.

Value

Nothing. The function produces the following maps:

- 'streams_v' derived streams with topology (vector)
- 'dirs' flow directions (raster)
- 'accums' accumulation values (raster)
- 'dem_cond' conditioned dem (raster) if `condition` is TRUE
- 'dem_[cond]_burn[X]' burnt in DEM (raster) if burn is > 0

The original GRASS map 'dem' is not modified if `condition` is TRUE and / or `burn` > 0.

Note

`setup_grass_environment` and `import_data` must be run before.

Author(s)

Mira Kattwinkel <`mira.kattwinkel@gmx.net`>, Eduard Szoecs, <`eduardszoecs@gmail.com`>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)
```

```
# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
streams_path <- system.file("extdata", "nc", "streams.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path, streams = streams_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 10, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Plot
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
sites <- readVECT('sites_o', ignore.stderr = TRUE)
streams_o <- readVECT('streams_o', ignore.stderr = TRUE)
streams <- readVECT('streams_v', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20))
lines(streams, col = 'blue', lwd = 2)
lines(streams_o, col = 'lightblue', lwd = 1)
points(sites, pch = 4)
```

`export_ssn` *Export 'SSN' object*

Description

This function exports the calculated sites, edges and binary IDs to a folder which then can be read using the 'SSN' package.

Usage

```
export_ssn(path, predictions = NULL, delete_directory = FALSE)
```

Arguments

`path`	character; path to write .ssn object to.
`predictions`	name(s) of prediction map(s) (optional).
`delete_directory`	boolean; shall the ssn directory and all files be deleted before export in case it already exists? See details.

Details

First it is checked if one of the column names is longer than 10 characters (which cannot be exported to ESRI shape files as required by 'SSN').

`delete_directory = TRUE` is useful if the same directory name has been used before and the existing data shall be overwritten.

Value

Nothing. Files are written to the specified folder

Author(s)

Mira Kattwinkel,<mira.kattwinkel@gmx.net>, Eduard Szoecs, <eduardszoecs@gmail.com>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Prepare site
calc_sites()
```

```
# Write data to SSN Folder
ssn_dir <- file.path(tempdir(), 'nc.ssn')
export_ssn(ssn_dir, delete_directory = TRUE)
list.files(ssn_dir)
```

`import_data` *Import data into 'GRASS.'*

Description

This function loads a DEM (digital elevation model) and sites data (both required) into the 'GRASS' session. Optionally, prediction sites and streams data can be loaded and the streams may be corrected by snapping to prevent lose ends. Likewise, potential predictor maps (raster or vector format) can be loaded.

Usage

```
import_data(
  dem,
  band = 1,
  sites,
  streams = NULL,
  snap_streams = FALSE,
  pred_sites = NULL,
  predictor_raster = NULL,
  predictor_r_names = NULL,
  predictor_vector = NULL,
  predictor_v_names = NULL
)
```

Arguments

`dem`	character; path to DEM (digital elevation model) raster file.
`band`	integer (optional); defines which band of the dem to use
`sites`	character string or object; path to sites vector file (ESRI shape) or sp or sf data object.
`streams`	character string or object (optional); path to network vector file (ESRI shape) or sp or sf data object. If available this can be burnt into the DEM in `derive_streams`
`snap_streams`	boolean (optional); snap line ends. If TRUE line ends of the streams are snapped to the next feature if they are unconnected with threshold of 10 m using 'GRASS' function v.clean.
`pred_sites`	character string vector or object(s) (optional); path(s) to prediction sites vector files (ESRI shape) or sp or sf data object. Different formats (i.e. path and objects) must not be mixed; more than one sf or sp object must be provided as a list, not concatenated with `c`.

`predictor_raster`
: character vector (optional); paths to raster data to import as predictors.

`predictor_r_names`
: character string vector (optional); names for potential predictor variables in raster format; if not provided `perdictor_raster` is used.

`predictor_vector`
: character string vector of object(s) (optional); path(s) to vector data (ESRI shape) or sp or sf object names to import as predictors. Different formats (i.e. path and objects) must not be mixed; more than one sf or sp object must be provided as a list, not concatenated with `c`.

`predictor_v_names`
: character vector (optional); names for potential predictor variables in vector format ; if not provided `perdictor_vector` is used.

Details

All vector data (sites, streams and potential predictors) is imported into the current location using v.import. Hence, if the projections does not match to the one of the DEM (which was used to specify the location in `setup_grass_environment`) the maps are projected and imported on the fly. All raster data are not transformed but it is assumed that they have the same projection as the current location. Hence, it is important to make sure that they all have indeed the same projection (and same cell size) and that the correct one is set in `setup_grass_environment`. If this condition is not met, the raster data should be propocessed before importing. Use `check_projection` to compare the projection of a raster data set and the one of the current location.

Value

Nothing, the data is loaded into the 'GRASS' session (mapset PERMANENT). The DEM is stored as raster 'dem', sites as vector 'sites_o', prediction sites as vector using the original file names with an appended '_o' (without extension), streams as vector 'streams_o' in the 'GRASS' location. Additionally, predictor raster map(s) can be read in and are stored in 'GRASS' using either the original file names (without extension) or using the names provides in `predictor_r_names`. The latter option may be useful if ArcGIS grid data (typically stored as 'grid_name/w001001.adf') are used. Likewise, predictor vector maps can be read in from Esri Shape file (given as the full file path) or as sf or sp objects. Potential predictor data can also be read in later, e.g. using GRASS commands v.import or r.in.gdal (see examples below).

Note

A GRASS session must be initiated before, see `initGRASS`.

If sites, pred_sites and / or streams are sp objects it is important that they have a datum defined otherwise the import will not work. Hence, it is e.g. better to use proj4string = CRS("+proj=tmerc +lat_0=0 +lon_0=9 +k=1 +x_0=3500000 +y_0=0 +datum=potsdam +units=m +no_defs") instead of proj4string = CRS("+proj=tmerc +lat_0=0 +lon_0=9 +k=1 +x_0=3500000 +y_0=0 +ellps=bessel +towgs84=598.1,73.7,418.2,0.202,0.045,-2.455,6.7 +units=m +no_defs")) when defining sp objects.

Author(s)

Eduard Szoecs, <`eduardszoecs@gmial.com`>, Mira Kattwinkel <`mira.kattwinkel@gmx.net`>

Examples

```
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
streams_path <- system.file("extdata", "nc", "streams.shp", package = "openSTARS")
preds_v_path <- system.file("extdata", "nc", "pointsources.shp", package = "openSTARS")
preds_r_path <- system.file("extdata", "nc", "landuse_r.tif", package = "openSTARS")

setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path, streams = streams_path,
            predictor_vector = preds_v_path, predictor_raster = preds_r_path)
gmeta()

# Plot data
library(sp)
dem <- readRAST("dem", ignore.stderr = TRUE)
sites_orig <-  readVECT("sites_o", ignore.stderr = TRUE)
lu <- readRAST("landuse_r", ignore.stderr = TRUE)
# import additional vector data
fp <-  system.file("extdata", "nc", "pointsources.shp", package = "openSTARS")
execGRASS("v.import", flags = c("overwrite", "quiet"),
parameters = list(
  input = fp,
  output =  "psources",
  extent = "region"),  # to import into current regien
  intern = TRUE, ignore.stderr = TRUE)

#plot(dem, col = terrain.colors(20))
plot(dem, col = grey.colors(20))
points(sites_orig, pch = 4)
ps <- readVECT("psources")
points(ps, bg = "red", pch = 21, col = "grey", cex = 1.5)

# plot landuse data
library(raster)
op <- par()
par(xpd = FALSE)
plot(raster(lu), legend = FALSE, xaxt = "n", yaxt = "n", bty = "n",
col = c("red", "goldenrod", "green", "forestgreen","darkgreen", "blue", "lightblue"))
par(xpd = TRUE)
legend("bottom", cex = 0.75,
  legend = c("developed", "agriculture", "herbaceous", "shrubland", "forest", "water", "sediment"),
   fill = c("red", "goldenrod", "green", "forestgreen","darkgreen", "blue", "lightblue"),
   horiz = TRUE, inset = -0.175)
par <- op
```

`import_vector_data` *Import vector data into GRASS.*

Description

Generic function to import vector data of various formats into the GRASS environment.

Usage

```
import_vector_data(data, name, layer = NULL, proj_ref_obj = NULL, snap = -1)
```

Arguments

`data`	character string or object; path to data vector file (shape), postgis data source name (dsn; see details), or sp or sf data object.
`name`	string giving the base name of the vector data within the GRASS envrionment (i.e. output)
`layer`	character string; default 1, particularly needed if data is a dsn for importing postgis data (see details)
`proj_ref_obj`	character; path to a georeferenced data file to be used as reference; only used if `data` is an sf of sp object, then, the two projections are compared to find the correct way for importing; typically the dem raster file used in this project.
`snap`	float; snapping threshold in map units. If != -1 (default) vertices are snapped to other vertices in this snapping distance during import. If used, the features are automatically cleaned afterwards (see GRASS tools v.import and v.clean)

Details

For importing data from Postgis, all data base credentials must be supplied in `data` and the correct `layer` and, if the table containing the polygons are in a specific schema also that one (see example)

Value

Nothing.

Author(s)

Mira Kattwinkel, `<mira.kattwinkel@gmx.net>`

Examples

```
# import data from Postgis
## Not run:
import_vector_data(data = "PG: 'pgname=postgit_DB', 'host=123.45.67.890',
'port='1234', 'user=username', 'password=password'",
 name = "forest", layer = "landuse_schema.forest")

## End(Not run)
```

`merge_sites_measurements` *Merge a table with measurements to the sites.*

Description

After all processing steps are done and before exporting as an SSN object measurements can be added to the site map. They can contain multiple parameters and repeated measurements at the same site.

Usage

```
merge_sites_measurements(measurements, site_id, all_sites = FALSE, ...)
```

Arguments

`measurements`	character string, data.table or data.frame object; path to table data containing the data or a data.table or data.frame object
`site_id`	character string; columns name that gives the unique name of the site. Must be identical in both the sites vector object and the table of measurements
`all_sites`	locical; should sites without measurments be preserved (default FALSE)
`...`	additional agruments to read.table in case `measuremtes` is a file path to table data; see `read.table` for details.

Details

Measurements are merged to the sites objects based on `site_id`. If there are repeated measurements, point features are dublicated and the 'pid' of the sites is updated accoringly to be unique.

Author(s)

Mira Kattwinkel `<mira.kattwinkel@gmx.net>`

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex junctions (there are no complex confluences in this
# example date set)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# Prepare edges
calc_edges()

# Prepare site
calc_sites()

# Plot data
library(sp)
dem <- readRAST('dem', ignore.stderr = TRUE)
edges <- readVECT('edges', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
sites_o <- readVECT('sites_o', ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20),axes = TRUE)
lines(edges, col = 'blue')
points(sites, pch = 4)
points(sites_o, pch = 1)
legend("topright", pch = c(1, 4), legend = c("original", "corrected"))
```

---

openSTARS *openSTARS: An Open Source Implementation of the 'ArcGIS' Toolbox 'STARS'.*

Description

openSTARS provides functions to prepare data so that it can be imported by the SSN package for spatial modelling on stream networks. 'GRASS GIS 7.0' (or greater) with installed addons r.stream.basins, r.stream.distance, r.stream.order, and r.hydrodem is needed.

obs_data.csv

Artificial observation data with arbitrary measurements.

elev_ned30m.tif

South-West Wake county National Elevation Data 30m.

sites_nc.shp

Arbitrary sites along rivers in North Carolina.

streams.shp

Rivers in North Carolina.

geology.shp

Geological data.

landuse_r.tif

Land use date in North Carolina.

lakes.shp

Artificial lakes (not at topologically correct locations)

pointsources.shp

Artificial point sources.

Examples

```
if(.Platform$OS.type == "windows"){
gisbase = "c:/Program Files/GRASS GIS 7.6"
} else {
  gisbase = "/usr/lib/grass74/"
}
initGRASS(gisBase = gisbase,
          home = tempdir(),
          override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
```

```
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
streams_path <- system.file("extdata", "nc", "streams.shp", package = "openSTARS")
preds_v_path <- system.file("extdata", "nc", "pointsources.shp", package = "openSTARS")
preds_r_path <- system.file("extdata", "nc", "landuse_r.tif", package = "openSTARS")


setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path, streams = streams_path,
            predictor_vector = preds_v_path, predictor_raster = preds_r_path)
gmeta()

# Derive streams from DEM
# burn in 'streams' 10 meters
derive_streams(burn = 10, accum_threshold = 700, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

# calculate slope as potential predictor
execGRASS("r.slope.aspect", flags = c("overwrite","quiet"),
          parameters = list(
            elevation = "dem",
            slope = "slope"
          ))

# Prepare edges
calc_edges()
calc_attributes_edges(input_raster = c("slope", "landuse_r"),
                      stat_rast = c("max", "percent"),
                      attr_name_rast = c("maxSlo", "luse"),
                      input_vector = "pointsources", stat_vect = "count",
                      attr_name_vect = "psource")

# Prepare site
calc_sites()

# Usually, only one of the following methods is needed. The exact one takes
# much longer to run
# approximate potential predictor variables for each site based on edge values
calc_attributes_sites_approx(input_attr_name = c("maxSlo", "lusep_1", "lusep_2",
                                                 "lusep_3", "lusep_4", "lusep_5",
                                                 "lusep_6", "lusep_7"),
                             output_attr_name = c("maxSloA","luse1A", "luse2A",
                                                  "luse_3A", "luse4A", "luse5A",
                                                  "luse6A", "luse7A"),
                             stat = c("max", rep("percent", 7)))

# exact potential predictor variables for each site based on catchments
```

```
calc_attributes_sites_exact(input_raster = c("slope", "landuse_r"),
                            attr_name_rast = c("maxSloEx", "luseE"),
                            stat_rast = c("max", "percent"))

# Plot data
library(sp)
dem <- readRAST("dem", ignore.stderr = TRUE)
sites <- readVECT("sites", ignore.stderr = TRUE)
sites_orig <-  readVECT("sites_o", ignore.stderr = TRUE)
edges <- readVECT("edges", ignore.stderr = TRUE)
plot(dem, col = terrain.colors(20))
lines(edges, col = "blue")
points(sites_orig, pch = 4)
cols <- colorRampPalette(c("blue", "red"))(length(sites$H2OArea))[rank(sites$H2OArea)]
points(sites, pch = 16, col = cols)

# Write data to SSN Folder
ssn_dir <- file.path(tempdir(), "nc.ssn")
export_ssn(ssn_dir, delete_directory = TRUE)

# Check if all files are ok
library(SSN)
check_ssn(ssn_dir)

# Load into SSN-package
ssn_obj <- importSSN(ssn_dir, o.write = TRUE)
print(ssn_obj)



#Datasets shipped with openSTARS
```

`prepare_sites` *Snap sites to streams and calculate attributes*

Description

Snap sites to streams and calculate attributes

Usage

```
prepare_sites(sites_map, locid_c = NULL, pid_c = NULL, maxdist = NULL)
```

Arguments

`sites_map`	character; name of sites map (observation or prediction sites) as created by `import_data`.
`locid_c`	character (optional); column name in the sites attribute table giving a unique site identifier.

`pid_c`	character (optional); column name in the sites attribute table that distinguishes between repeated measurements at a sampling site.
`maxdist`	integer (optional); maximum snapping distance. Sites farther away from edges will be deleted.

Details

This function is called by `calc_sites` and should not be called directly. Sites are snapped to the streams and upstream distance is calculated.

`restrict_network` *Restrict edges to certain stream networks*

Description

When the stream network is derived from a DEM, the network will cover the whole extent of the DEM input. However, the obervation sites might be restricted to a certain area, i.e. to certain networks. This functin deletes edges that belong to networks (based on their netID) without sites (observation or prediction). This can help to save computation time before calculating edge attributes.

Usage

```
restrict_network(
  sites = NULL,
  keep_netIDs = NULL,
  delete_netIDs = NULL,
  keep = TRUE,
  filename = "edges_o"
)
```

Arguments

`sites`	name(s) of sites.
`keep_netIDs`	numeric (optional); vector of netIDs to keep
`delete_netIDs`	numeric (optional); vector of netIDs to delete
`keep`	boolean; should the original 'edges' be saved? Default is TRUE.
`filename`	character string; file name to save the original edges vector file; defaults to 'edges_o'.

Value

Nothing. The function updates 'edges' and (if keep = TRUE) saves the original file to the file name provided.

Author(s)

Mira Kattwinkel <mira.kattwinkel@gmx.net>

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
    gisbase <- "c:/Program Files/GRASS GIS 7.6"
} else {
  gisbase <- "/usr/lib/grass74/"
}
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)

# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
sites_path <- system.file("extdata", "nc", "sites_nc.shp", package = "openSTARS")
lakes_path <- system.file("extdata", "nc", "lakes.shp", package = "openSTARS")
setup_grass_environment(dem = dem_path)
import_data(dem = dem_path, sites = sites_path)
gmeta()

# Derive streams from DEM
derive_streams(burn = 0, accum_threshold = 100, condition = TRUE, clean = TRUE)

# Check and correct complex confluences (there are no complex confluences in this
# example date set; set accum_threshold in derive_streams to a smaller value
# to create complex confluences)
cj <- check_compl_confluences()
if(cj){
  correct_compl_confluences()
}

calc_edges()
calc_sites()
restrict_network(sites = "sites", keep = TRUE, filename = "edges_o")

# plot
library(sp)
edges <- readVECT('edges', ignore.stderr = TRUE)
edges_o <- readVECT('edges_o', ignore.stderr = TRUE)
sites <- readVECT('sites', ignore.stderr = TRUE)
plot(edges_o, col = "lightblue", lwd = 2)
lines(edges, col = "blue4")
points(sites, pch = 16, col = "red")
legend("topright", col = c("red", "lightblue", "blue4"), lty = c(NA, 1,1),
lwd = c(NA,2,1), pch = c(16,NA,NA),
legend = c("sites", "edges original", "edges restricted"))
```

setup_grass_environment *Setup 'GRASS' environment.*

Description

This function sets the 'GRASS' mapset to PERMANENT and sets its projection and extension.

Usage

```
setup_grass_environment(dem, epsg = NULL, sites = NULL)
```

Arguments

`dem`	character; path to DEM.
`epsg`	integer (optional); EPSG code for the projection to use. If not given (default) the information is taken from the dem. This should ONLY be used if the `dem` does not contain projection information and MUST be the correct one for the dem used.
`sites`	(deprecated); not used any more. Only included for compatibility with previous version.

Value

Nothing, the 'GRASS' mapset is set to PERMANENT, the extent of the current location is set to the one of the dem, and the projection is set to the one of the dem or to the one provided in `epsg` .

Note

A 'GRASS' session must be initiated before, see `initGRASS`. This function uses `use_sp()` from the `rgrass7` package to support raster data.

Author(s)

Mira Kattwinkel, `<mira.kattwinkel@gmx.net>`

Examples

```
# Initiate GRASS session
if(.Platform$OS.type == "windows"){
  gisbase = "c:/Program Files/GRASS GIS 7.6"
  } else {
  gisbase = "/usr/lib/grass74/"
  }
initGRASS(gisBase = gisbase,
    home = tempdir(),
    override = TRUE)
```

```
# Load files into GRASS
dem_path <- system.file("extdata", "nc", "elev_ned_30m.tif", package = "openSTARS")
setup_grass_environment(dem = dem_path)
gmeta()
```

Index