Package: dynatopGIS (via r-universe)

July 6, 2024

Title Algorithms for Helping Build Dynamic TOPMODEL Implementations from Spatial Data

Version 0.3.0.1010

Description A set of algorithms based on Quinn et al. (1991) <doi:10.1002/hyp.3360050106> for processing river network and digital elevation data to build implementations of Dynamic TOPMODEL, a semi-distributed hydrological model proposed in Beven and Freer (2001) <doi:10.1002/hyp.252>. The 'dynatop' package implements simulation code for Dynamic TOPMODEL based on the output of 'dynatopGIS'.

License GPL-2

Encoding UTF-8

LazyData true

Imports R6, terra, jsonlite

Depends R (>= 4.0.0)

BugReports https://github.com/waternumbers/dynatopGIS/issues

URL https://waternumbers.github.io/dynatopGIS/, https://github.com/waternumbers/dynatopGIS

RoxygenNote 7.3.1

Suggests knitr, rmarkdown, tinytest

VignetteBuilder knitr

Language en-GB

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

RemoteUrl https://github.com/waternumbers/dynatopgis

RemoteRef HEAD

RemoteSha f4f59c639eeb43f9cd6a229164bb28dd0c824d9d

Contents

`convert_channel` *Function for assisting in the conversion of object to be suitable channel inputs to a dynatopGIS object*

Description

Converts SpatialLinesDataFrame or SpatialPolygonsDataFrame to the correct format of SpatialPolygonsDataFrame for dynatopGIS.

Usage

```
convert_channel(
  vect_object,
 property_names = c(name = "DRN_ID", length = "length", startNode = "startNode", endNode
    = "endNode", width = "width", slope = "slope"),
  default_width = 2,
  default_slope = 0.001
)
```

Arguments

`vect_object`	a SpatVect object or a file which can read by terra::vect to create one
`property_names`	a named vector of containing the columns of existing data properties required in the final SpatialPolygonsDataFrame
`default_width`	the width in m to be used for buffering lines to produce polygons
`default_slope`	the slope in m/m to be used when none is provided

Details

If the property_names vector contains a width this is used for buffering lines to produce polygons, otherwise the default_width value is used.

Examples

```
channel_file <- system.file("extdata", "SwindaleRiverNetwork.shp",
package="dynatopGIS", mustWork = TRUE)
vect_lines <- terra::vect(channel_file)
property_names <- c(name="identifier",endNode="endNode",startNode="startNode",length="length")
chn <- convert_channel(vect_lines,property_names)
```

dynatopGIS *R6 Class for processing a catchment to make a Dynamic TOPMODEL*

Description

R6 Class for processing a catchment to make a Dynamic TOPMODEL

R6 Class for processing a catchment to make a Dynamic TOPMODEL

Methods

Public methods:

- dynatopGIS$new()
- dynatopGIS$add_catchment()
- dynatopGIS$add_dem()
- dynatopGIS$add_channel()
- dynatopGIS$add_layer()
- dynatopGIS$get_layer()
- dynatopGIS$plot_layer()
- dynatopGIS$sink_fill()
- dynatopGIS$compute_band()
- dynatopGIS$compute_properties()
- dynatopGIS$compute_flow_lengths()
- dynatopGIS$classify()
- dynatopGIS$combine_classes()
- dynatopGIS$create_model()
- dynatopGIS$get_version()
- dynatopGIS$get_method()
- dynatopGIS$clone()

Method `new()`**:** Initialise a project, or reopen an existing project

Usage:

```
dynatopGIS$new(projectFolder)
```

Arguments:

`projectFolder` folder for data files

Details: This loads the project data files found in the `projectFolder` if present. If not the folder is created. The project data files are given by `projectFolder/<filename>.<tif,shp>`

Returns: A new 'dynatopGIS' object

Method `add_catchment()`**:** Add a catchment outline to the 'dynatopGIS' project

Usage:

```
dynatopGIS$add_catchment(catchment)
```

Arguments:

`catchment` a `SpatRaster` object or the path to file containing one which contains a rasterised catchment map.

Details: If not a `SpatRaster` object the the catchment is read in using the terra package. Finite values in the raster indicate that the area is part of the catchment; with each subcatchment taking a unique finite value. Note that in the later processing it is assumed that outflow from the subcatchments can occur only through the channel network. The resolution and projection of the project is taken from the provided catchment

Returns: `invisible(self)`

Method `add_dem()`**:** Import a dem to the 'dynatopGIS' object

Usage:

```
dynatopGIS$add_dem(dem, fill_na = -9999)
```

Arguments:

`dem` a `raster` layer object or the path to file containing one which is the DEM

`fill_na` should NA values in dem be filled. See details

`verbose` Should additional progress information be printed

Details: If not a `raster` the DEM is read in using the terra package. If `fill_na` is `TRUE` all NA values other then those that link to the edge of the dem are filled so they can be identified as sinks.

Returns: suitable for chaining

Method `add_channel()`**:** Import channel data to the 'dynatopGIS' object

Usage:

```
dynatopGIS$add_channel(channel, verbose = FALSE)
```

Arguments:

`channel` a SpatVect object or file path that can be loaded as one containing the channel information

`verbose` Should additional progress information be printed

Details: Takes the representation of the channel network as a SpatVect with properties name, length, area, startNode, endNode and overlaying it on the DEM. In doing this a variable called id is created (or overwritten) other variables in the data frame are passed through unaltered.

Returns: suitable for chaining

Method `add_layer()`**:** Add a layer of geographical information

Usage:

```
dynatopGIS$add_layer(layer, layer_name = names(layer))
```

Arguments:

`layer` the raster layer to add (see details)

`layer_name` name to give to the layer

Details: The layer should either be a raster layer or a file that can be read by the `raster` package. The projection, resolution and extent are checked against the existing project data. Only layer names not already in use (or reserved) are allowed. If successful the layer is added to the project tif file.

Returns: suitable for chaining

Method `get_layer()`**:** Get a layer of geographical information or a list of layer names

Usage:

```
dynatopGIS$get_layer(layer_name = character(0))
```

Arguments:

`layer_name` name of the layer give to the layer

Returns: a 'raster' layer of the requested information if layer_name is given else a vector of layer names

Method `plot_layer()`**:** Plot a layer

Usage:

```
dynatopGIS$plot_layer(layer_name, add_channel = TRUE)
```

Arguments:

`layer_name` the name of layer to plot

`add_channel` should the channel be added to the plot

Returns: a plot

Method `sink_fill()`**:** The sink filling algorithm of Planchona and Darboux (2001)

Usage:

```
dynatopGIS$sink_fill(
  min_grad = 1e-04,
  max_it = 1e+06,
  verbose = FALSE,
  hot_start = FALSE,
  flow_type = c("quinn", "d8")
)
```

Arguments:

`min_grad` Minimum gradient between cell centres

`max_it` maximum number of replacement cycles

`verbose` print out additional diagnostic information

`hot_start` start from filled_dem if it exists

`flow_type` The type of flow routing to apply see details

Details: The algorithm implemented is based on that described in Planchona and Darboux, "A fast, simple and versatile algorithm to fill the depressions in digital elevation models" Catena 46 (2001). A pdf can be found at (<https://horizon.documentation.ird.fr/exl-doc/pleins_textes/pleins_textes_7/sous_copyrigh
The adaptations made are to ensure that all cells drain only within the subcatchments if provided. The flow_type can be either - "quinn" where flow is split across all downslope directions or - "d8" where all flow follows the steepest between cell gradient

Method `compute_band()`**:** Computes the computational band of each cell

Usage:

```
dynatopGIS$compute_band(type = c("strict"), verbose = FALSE)
```

Arguments:

`type` type of banding

`verbose` print out additional diagnostic information

Details: Banding is used within the model to define the HRUs and control the order of the flow between them; HRUs can only pass flow to HRUs in a lower numbered band. Currently only a strict ordering of river channels and cells in the DEM is implemented. To compute this the algorithm passes first up the channel network (with outlets being in band 1) then through the cells of the DEM in increasing height.

Method `compute_properties()`**:** Computes statistics e.g. gradient, log(upslope area / gradient) for raster cells

Usage:

```
dynatopGIS$compute_properties(min_grad = 1e-04, verbose = FALSE)
```

Arguments:

`min_grad` gradient that can be assigned to a pixel if it can't be computed

`verbose` print out additional diagnostic information

Details: The algorithm passed through the cells in decreasing height. Min grad is applied to all cells. It is also used for missing gradients in pixels which are partially channel but have no upslope neighbours.

Method `compute_flow_lengths()`**:** Computes flow length for each pixel to the channel

Usage:

```
dynatopGIS$compute_flow_lengths(
  flow_routing = c("expected", "dominant", "shortest"),
  verbose = FALSE
)
```

Arguments:

`flow_routing` TODO

`verbose` print out additional diagnostic information

Details: The algorithm passes through the cells in the DEM in increasing height. Three measures of flow length to the channel are computed. The shortest length (minimum length to channel through any flow path), the dominant length (the length taking the flow direction with the highest fraction for each pixel on the path) and expected flow length (flow length based on sum of downslope flow lengths based on fraction of flow to each cell). By definition cells in the channel that have no land area have a length of NA.

Method `classify()`**:** Create a catchment classification based cutting an existing layer into classes

Usage:

```
dynatopGIS$classify(layer_name, base_layer, cuts)
```

Arguments:

`layer_name` name of the new layer to create

`base_layer` name of the layer to be cut into classes

`cuts` values on which to cut into classes. These should be numeric and define either the number of bands (single value) or breaks between band (multiple values).

Details: This applies the given cuts to the supplied landscape layer to produce areal groupings of the catchment. Cuts are implement using `terra::cut` with `include.lowest = TRUE`. Note that is specifying a vector of cuts values outside the limits will be set to NA.

Method `combine_classes()`**:** Combine any number of classifications based on unique combinations and burns

Usage:

```
dynatopGIS$combine_classes(layer_name, pairs, burns = NULL)
```

Arguments:

`layer_name` name of the new layer to create

`pairs` a vector of layer names to combine into new classes through unique combinations. Names should correspond to raster layers in the project directory.

`burns` a vector of layer names which are to be burnt on

Details: This applies the given cuts to the supplied landscape layers to produce areal groupings of the catchment. Burns are added directly in the order they are given. Cuts are implement using `terra::cut` with `include.lowest = TRUE`. Note that is specifying a vector of cuts values outside the limits will be set to NA.

Method `create_model()`**:** Compute a Dynamic TOPMODEL

Usage:

```
dynatopGIS$create_model(
  layer_name,
  class_layer,
  sf_opt = c("cnst", "kin"),
  sz_opt = c("exp", "bexp", "cnst", "dexp"),
  rain_layer = NULL,
  rain_label = character(0),
  pet_layer = NULL,
  pet_label = character(0),
  verbose = FALSE
)
```

Arguments:

`layer_name` name for the new model and layers

`class_layer` the layer defining the topographic classes

`sf_opt` Surface solution to use

`sz_opt` transmissivity profile to use

`rain_layer` the layer defining the rainfall inputs

`rain_label` Prepended to rain_layer values to give rainfall series name

`pet_layer` the layer defining the pet inputs

`pet_label` Prepended to pet_layer values to give pet series name

`verbose` print more details of progress

Details: The `class_layer` is used to define the HRUs. Flow between HRUs is based on the ordering of the catchment (see the `compute_band` method). Flow from a HRU can only go to a HRU with a lower band. Setting the sf_opt and sz_opt options ensures the model is set up with the correct parameters present. The `rain_layer` (`pet_layer`) can contain the numeric id values of different rainfall (pet) series. If the value of `rain_layer` (`pet_layer`) is not `NULL` the weights used to compute an averaged input value for each HRU are computed, otherwise an input table for the models generated with the value "missing" used in place of the series name.

Method `get_version()`**:** get the version number

Usage:

```
dynatopGIS$get_version()
```

Details: the version number indicates the version of the algorithms within the object

Returns: a numeric version number

Method `get_method()`**:** get the cuts and burns used to classify

Usage:

```
dynatopGIS$get_method(layer_name)
```

Arguments:

`layer_name` the name of layer whose classification method is returned

Returns: a list with two elements, cuts and burns

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
dynatopGIS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## The vignettes contains more examples of the method calls.

## create temporary directory for output
demo_dir <- tempfile("dygis")
dir.create(demo_dir)

## initialise processing
ctch <- dynatopGIS$new(file.path(demo_dir,"test"))

## add a catchment outline based on the digital elevation model
dem_file <- system.file("extdata", "SwindaleDTM40m.tif", package="dynatopGIS", mustWork = TRUE)
dem <- terra::rast(dem_file)
dem <- terra::extend(dem,1)
catchment_outline <- terra::ifel(is.finite(dem),1,NA)
ctch$add_catchment(catchment_outline)

## add digital elevation and channel data
ctch$add_dem(dem)
```

```
channel_file <- system.file("extdata", "SwindaleRiverNetwork.shp",
package="dynatopGIS", mustWork = TRUE)
sp_lines <- terra::vect(channel_file)
property_names <- c(name="identifier",endNode="endNode",startNode="startNode",length="length")
chn <- convert_channel(sp_lines,property_names)
ctch$add_channel(chn)

## compute properties
ctch$sink_fill() ## fill sinks in the catchment and computes dem flow directions

ctch$compute_properties() # like topograpihc index and contour length
ctch$compute_band()
ctch$compute_flow_lengths()

## classify and create a model

ctch$classify("atb_20","atb",cuts=20) # classify using the topographic index
ctch$get_method("atb_20") ## see the details of the classification
ctch$combine_classes("atb_20_band",c("atb_20","band")) ## combine classes
ctch$create_model(file.path(demo_dir,"new_model"),"atb_20_band") ## create a model
list.files(demo_dir,pattern="new_model*") ## look at the output files for the model

## tidy up
unlink(demo_dir)
```

Index