

Package: dynatop (via r-universe)

October 1, 2026

Title An Implementation of Dynamic TOPMODEL Hydrological Model in R

Version 0.5.0.4

Description An R implementation and enhancement of the Dynamic TOPMODEL semi-distributed hydrological model originally proposed by Beven and Freer (2001) <[doi:10.1002/hyp.252](https://doi.org/10.1002/hyp.252)>. The 'dynatop' package implements code for building and simulating models.

License AGPL-3

Encoding UTF-8

LazyData true

ByteCompile true

Imports R6, Rcpp, terra, jsonlite, methods, RcppParallel

LinkingTo Rcpp, RcppParallel

Depends R (>= 4.0.0)

BugReports <https://codeberg.org/dynatop/dynatop/issues>

URL <https://codeberg.org/dynatop/dynatop>,
<https://dynatop.codeberg.page>

Suggests rmarkdown, knitr, tinytest

SystemRequirements GNU make

VignetteBuilder knitr

Language en-GB

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libgdal-dev gdal-bin libgeos-dev make libproj-dev libsqlite3-dev

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

Date/Publication 2026-10-01 08:55:42 UTC

RemoteUrl <https://codeberg.org/dynatop/dynatop>

RemoteRef HEAD

RemoteSha e0502f62c614b9a938f2b2743212e1939b852bf8

Contents

check_channel	2
check_water_bodies	3
convert_channel	4
daily_evap_profile	5
dynatop	7
dynatopGIS	9
file_functions	14
fill_na	14
simplify_channel	15
trim_channel	16
Index	18

check_channel	<i>Function for checking channel networks are valid and suitable for dynatopGIS</i>
---------------	---

Description

Performs checks on the values and properties of a channel then checks connnections to outlets if given

Usage

```
check_channel(chn, outlets = NULL, check_connectivity = TRUE)
```

Arguments

- chn a SpatVect object representing the channel newtork as returned by convert_channel
- outlets uid values of the outlets of the channel network
- check_connectivity check the connectivity of the channel network

Details

The processing follows the follwoing sequence: - Checks of the required properties of the channel netowrk for use in dynatopGIS - checks on the connectivity for loops - If outlets provided comparision of identified and known outlets
depth and width are only checked is present

Value

TRUE if completes else failure message

Examples

```

chnFile <- system.file("extdata", "gis", "SwindaleRiverNetwork.gpkg",
                        package="dynatop", mustWork = TRUE)
chn <- terra::vect(chnFile)
chn$slope <- 1e-3
chn$class <- 1
chn$width <- 1
chn$depth <- 0.1
chn <- convert_channel(chn,
                      property_names=c(uid = "identifier",
                                       startNode = "startNode",
                                       endNode = "endNode",
                                       slope = "slope",
                                       class = "class",
                                       width="width",
                                       depth="depth"))

check_channel(chn)

```

check_water_bodies	<i>Function for checking water bodies are valid and suitable for dynatopGIS</i>
--------------------	---

Description

Performs checks on the values and properties of a water body

Usage

```
check_water_bodies(wb)
```

Arguments

wb a SpatVect object representing the water bodies.

Details

The processing checks of the required properties of the water body for use in dynatopGIS: - uid: unique character uid - class: numeric - slope: positive numeric - width: positive numeric - depth: positive numeric

Value

TRUE if completes else failure message

Examples

```
wbFile <- system.file("extdata", "gis", "SwindaleRiverNetwork.gpkg",
                      package="dynatop", mustWork = TRUE)
wb <- terra::vect(wbFile)
wb <- terra::buffer(wb,5)
wb$uid <- paste0("wb",seq_len(nrow(wb)))
wb$slope <- 1e-3
wb$class <- 1
wb$width <- 1
wb$depth <- 0.1
check_water_bodies(wb)
```

convert_channel	<i>Function for assisting in the conversion of object to be suitable channel inputs to a dynatopGIS object</i>
-----------------	--

Description

Converts SpatVect of the channel network into SpatVect containing polygons suitable for dynatopGIS

Usage

```
convert_channel(
  chn,
  property_names = c(uid = "uid", startNode = "startNode", endNode = "endNode", slope =
    "slope", class = "class", width = "width", depth = "depth"),
  min_slope = 0.001
)
```

Arguments

chn	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
min_slope	the minimum slope

Details

The processing follows the following sequence: - The 'property_names' input is used to rename the data associated with spatial objects. Remaining data is dropped - Variables are converted to the expected type then checked for missing values; in doing this - Missing or lower 'slope' values are populated by the min_slope and a warning issued

Value

A SpatVect containing polygons of the channel network, with at least the following properties: uid, width, depth, slope, startNode and endNode.

Examples

```
chnFile <- system.file("extdata", "gis", "SwindaleRiverNetwork.gpkg",
                       package="dynatop", mustWork = TRUE)
chn <- terra::vect(chnFile)
chn$slope <- 1e-3
chn$class <- 1
chn$width <- 1
chn$depth <- 0.1
chn <- convert_channel(chn,
                      property_names=c(uid = "identifier",
                                       startNode = "startNode",
                                       endNode = "endNode",
                                       slope = "slope",
                                       class = "class",
                                       width="width",
                                       depth="depth"))
```

daily_evap_profile	<i>Create sinusoidal time series of potential evapotranspiration input</i>
--------------------	--

Description

Two functions to generate daily profile properties then apply these to a time series

Usage

```
daily_evap_profile(eMin = 0, eMax = 3, max_day = 183)

evap_est(ts, daily_profile, daily_data = NULL, pet_name = "pet")
```

Arguments

eMin	Minimum daily PE total (m or mm)
eMax	Maximum daily PE total (m or mm)
max_day	Year day of maximum PET
ts	as vector of POSIXct data/times
daily_profile	data.frame containing annual profile properties for PET calculation
daily_data	data frame containing daily PET values for disaggregation
pet_name	column of daily_data containing the PET values

Details

Dynamic TOPMODEL requires a time series of potential evapotranspiration in order to calculate and remove actual evapotranspiration from the root zone during a run. Many sophisticated physical models have been developed for estimating potential and actual evapotranspiration, including the Priestley-Taylor (Priestley and Taylor, 1972) and Penman-Monteith (Montieth, 1965) methods.

These, however, require detailed meteorological data such as radiation input and relative humidities that are, in general, difficult to obtain. Calder (1983) demonstrated that a simple approximation using a sinusoidal variation in potential evapotranspiration to be a good approximation to more complex schemes.

If the insolation is also taken to vary sinusoidally through the daylight hours then, ignoring diurnal meteorological variations, the potential evapotranspiration during daylight hours for each year day number can be calculated (for the catchment's latitude). Integration over the daylight hours allows the daily maximum to be calculated and thus a sub-daily series generated.

The `daily_evap_profile` function computes a profile properties based on a simple sinusoid giving year day values for PET, dawn and day length. The `evap_est` function takes a time series of steps and computes the aggregated PET over each time step based on the profile and optionally daily PET values.

All inputs times are handled as though they are UTC!

Value

`daily_evap_profile` returns a `data.frame` of year day, dawn (s since start of day), day length (s) and PET (units as `eMin/eMax`). `evap_est` a `data.frame` with columns `dateTime` containing the time stamps of the data and `pet` containing the PET values.

References

Beven, K. J. (2012). Rainfall-runoff modelling : the primer. Chichester, UK, Wiley-Blackwell.

Calder, I. R., Harding, R. J. and Rosier, P. T. W. (1983) An objective assessment of soil-moisture deficit models. *Journal of Hydrology*, 60(1-4) 329-355

Examples

```
## Generating daily and hourly PET data for 1970
## the values of eMin and eMax may not be realistic
st_day <- as.POSIXct("1970-01-02 00:00:00",tz='GMT')
st_hour <- as.POSIXct("1970-01-01 01:00:00",tz='GMT')
fn <- as.POSIXct("1971-01-01 00:00:00",tz='GMT')

daily_prof <- daily_evap_profile(0,1)
daily_ts <- seq(st_day,fn,by=24*60*60)
daily_pet <- evap_est(daily_ts,daily_prof)

hourly_ts <- seq(st_hour,fn,by=1*60*60)
hourly_pet <- evap_est(hourly_ts,daily_prof)

## the totals should be the same...
stopifnot(all.equal(sum(hourly_pet$pet), sum(daily_pet$pet)))
```

dynatop

R6 Class for Dynamic TOPMODEL

Description

R6 Class for Dynamic TOPMODEL

Methods

Public methods:

- `dynatop$new()`
- `dynatop$initialise()`
- `dynatop$sim()`
- `dynatop$get_states()`
- `dynatop$clone()`

`dynatop$new()`: Creates a dynatop class object from the a list based model description as generated by dynatopGIS.

Usage:

```
dynatop$new(geo, param, delta = 1e-13)
```

Arguments:

`geo` a data.frame describing the HRUs geography or path to a gpkg file that can read to create one

`param` a data.frame describing the parameters for each HRU class or path to a jsonfile that can read to create one

`delta` error allowed in checking redistribution sums

Details: This function makes some basic consistency checks on a list representing a dynamic TOPMODEL model. The checks performed and basic 'sanity' checks. They do not check for the logic of the parameter values nor the consistency of states and parameters. Sums of the redistribution matrices are checked to be in the range 1 +/- delta.

Returns: invisible(self) suitable for chaining

`dynatop$initialise()`: Initialises a dynatop object in the simplest way possible.

Usage:

```
dynatop$initialise(initial_states = NULL)
```

Arguments:

`initial_states` a date frame of states

Details: if `initial_states` are present these are used for the HRUs given. Otherwise a steady state initialisation is performed.

Returns: invisible(self) suitable for chaining

`dynatop$sim()`: Simulate the dynatop model

Usage:

```

dynatop$sim(
  obs_data,
  output_defn,
  keep_states = NULL,
  output_dir = NULL,
  sub_step = NULL,
  n_thread = 1
)

```

Arguments:

obs_data data.frame of observations

output_defn definition of the outputs from the model

keep_states a vector of POSIXct objects (e.g. from xts) giving the time stamp at which the states should be kept

output_dir the directory to write output to

sub_step simulation timestep in seconds, default value of NULL results in data time step

n_thread number of parallel threads to use in execution

Details: Saving the states at every timestep and keeping the mass balance can generate very large data sets!!

Returns: invisible(self) for chaining

`dynatop$get_states()`: Get the current states

Usage:

```

dynatop$get_states()

```

Returns: data.frame of current statea

`dynatop$clone()`: The objects of this class are cloneable with this method.

Usage:

```

dynatop$clone(deep = FALSE)

```

Arguments:

deep Whether to make a deep clone.

Examples

```

## the vignettes contains further details of the method calls.
mdlFile <- system.file("extdata", "mdl", "test_mdl.gpkg", package="dynatop", mustWork=TRUE)
paramFile <- system.file("extdata", "mdl", "test_mdl_param.json", package="dynatop", mustWork=TRUE)
obsFile <- system.file("extdata", "mdl", "SwindaleObs.csv", package="dynatop", mustWork=TRUE)
## create and initialise model
dyn <- dynatop$new(mdlFile, paramFile)
dyn$initialise() ## this initialisation isn't good - see the vignette
## define an output definition
defn <- data.frame(name="outlet", hru=1, scale=1, flux="q_sf")
## read in observed
obs <- read_obs(obsFile)
## simulate

```

```
out <- dyn$sim(obs,defn)
```

dynatopGIS

R6 Class for processing a catchment to make a dynatop TOPMODEL

Description

R6 Class for processing a catchment to make a dynatop TOPMODEL

Methods

Public methods:

- `dynatopGIS$new()`
- `dynatopGIS$add_catchment()`
- `dynatopGIS$add_channel()`
- `dynatopGIS$add_layer()`
- `dynatopGIS$get_layer()`
- `dynatopGIS$plot_layer()`
- `dynatopGIS$sink_fill()`
- `dynatopGIS$order()`
- `dynatopGIS$combine_layers()`
- `dynatopGIS$create_model()`
- `dynatopGIS$get_version()`
- `dynatopGIS$clone()`

`dynatopGIS$new()`: Initialise a project, or reopen an existing project

Usage:

```
dynatopGIS$new(projectDir)
```

Arguments:

`projectDir` is the directory of project data

Details: If the directory exists the existing tif files are loaded as the project.

Returns: A new 'dynatopGIS' object

`dynatopGIS$add_catchment()`: Add a catchment outline and dem to the 'dynatopGIS' project

Usage:

```
dynatopGIS$add_catchment(catchment, dem, fill_na = -9999)
```

Arguments:

`catchment` a SpatVect object or the path to file containing one which contains the catchment outline

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

`fill_na` value used to replace missing dem vales within the catchment

Details: If the inputs are not spatial objects they are read in using the terra package. The resolution and projection of the project is taken from the provided dem. All NA values in the dem within the catchment are replaced by the `fill_na` values so they can be identified as sinks.

Returns: `invisible(self)`

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

Usage:

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

Arguments:

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

`water_bodies` a `SpatVect` object of polygons 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`, `startNode`, `endNode`, `slope`, `depth` and `width`. This is rasterised. The polygons in `water_bodies` are then burnt in. it is presumed that each polygon has only one outlet.

Returns: suitable for chaining

`dynatopGIS$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 into a `SpatRast` by the terra 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.

Returns: suitable for chaining

`dynatopGIS$get_layer()`: Get a layer of geographical information or a list of layer names

Usage:

```
dynatopGIS$get_layer(layer_name = NULL)
```

Arguments:

`layer_name` name of the layer give to the layer or `NULL` to return list of layers

Returns: a `SpatRaster` containing the requested layer or a vector of layer names

`dynatopGIS$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

`dynatopGIS$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
)
```

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

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_copyright) The flow_type can be either MFD algorithm in TODO

`dynatopGIS$order()`: Computes the order (banding) to evaluate cells in

Usage:

```
dynatopGIS$order(verbose = FALSE)
```

Arguments:

verbose print out additional diagnostic information

Details: The algorithm passed through the cells to compute the band starting at the river outlets hru numbers are assigned in approxamtly band order

`dynatopGIS$combine_layers()`: Combine any number of layers into a unique classification

Usage:

```
dynatopGIS$combine_layers(class_name, layers)
```

Arguments:

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

Details: This combines the layers into unique classess and stores a raster of outcomes along with a json file of the classifications

`dynatopGIS$create_model()`: Create a Dynamic TOPMODEL

Usage:

```

dynatopGIS$create_model(
  model_name,
  class_layer = NULL,
  precip_layer = NULL,
  precip_label = "precip",
  pet_layer = NULL,
  pet_label = "pet",
  verbose = FALSE
)

```

Arguments:

`model_name` base of the name for the new model files
`class_layer` the layer defining the landscape classes
`precip_layer` the layer defining the rainfall inputs
`precip_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 assign classes to the HRUs. Flow between HRUs are based on the ordering of the catchment (see the `order` method) with low from a HRU can only go to a HRU with a lower band. 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 value for the precipitation and PET inputs for each HRU are taken as e.g. `paste0(precip_label,precip)` otherwise the value of e.g. `precip_label` is used. The `model_name` can contain the path to the files.

Returns: Two files `model_name.gpkg` which contains the spatial description of the model and `model_name_param.json` an example parameter file for the model.

`dynatopGIS$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

`dynatopGIS$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)

## locate the dem, channel and catchment outline files
chnFile <- system.file("extdata", "gis", "SwindaleRiverNetwork.gpkg",
                        package="dynatop", mustWork = TRUE)
demFile <- system.file("extdata", "gis", "SwindaleDTM.tif",
                        package="dynatop", mustWork = TRUE)
ctchFile <- system.file("extdata", "gis", "SwindaleBoundary.gpkg",
                        package="dynatop", mustWork = TRUE)

## Preprocess channel data
chn <- terra::vect(chnFile)
chn$slope <- 1e-3
chn$class <- 1
chn$width <- 1
chn$depth <- 0.1
chn <- convert_channel(chn,
                      property_names=c(uid = "identifier",
                                       startNode = "startNode",
                                       endNode = "endNode",
                                       slope = "slope",
                                       class = "class",
                                       width="width",
                                       depth="depth"))

chn <- simplify_channel(chn)

## create a project
dy <- dynatopGIS$new(demo_dir)
## add dem and catchment data
dy$add_catchment(ctchFile, demFile)
## add the channel
dy$add_channel(chn)
## fill the sinks
dy$sink_fill()
## compute the order
dy$order()
## add a new layer
tmp <- dy$get_layer("filled_dem")
tmp <- terra::classify(tmp,5)
names(tmp) <- "alt_band"
dy$add_layer(tmp)
## classify the layer
dy$combine_layers("class_alt_band","alt_band")
## build a model

dy$create_model(file.path(demo_dir,"new_model"),"class_alt_band") ## create a model
list.files(demo_dir,pattern="new_model*") ## look at the output files for the model

## tidy up
unlink(demo_dir)

```

file_functions	<i>Helper components for reading dynatop models</i>
----------------	---

Description

Method for reading both model maps and parameter files.

Usage

```
read_param(file_name)

write_param(param, file_name)

read_geo(file_name, verbose = FALSE)

write_geo(geo, file_name)

read_obs(file_name, time_zone = "UTC")
```

Arguments

file_name	name of file to read or write into
param	parameter file containing classes
verbose	provide details of progress
geo	geography object describing the catchment with locations, hru numbers,connectivity, classes etc.
time_zone	time zone of data

Details

WARNING: There is no checking of format

fill_na	<i>Fill NA values in a SpatRast object</i>
---------	--

Description

Fill NA values in a SpatRast object

Usage

```
fill_na(rst, mask = NULL, layer_names = names(rst))
```



```

                                endNode = "endNode",
                                slope = "slope",
                                class = "class",
                                width="width",
                                depth="depth"))

tchn <- simplify_channel(chn)
nrow(chn)
nrow(tchn)

```

trim_channel

Trim channel objects

Description

Trim channel objects to those draining to named outlets

Usage

```
trim_channel(chn, outlets, removed = FALSE)
```

Arguments

chn	the channel object
outlets	names of the outlets to consider
removed	should the removed objects be returned

Details

Starting with the named outlet objects the channel network is passed up using the connectivity given by the start and end node, only those flagged as connected are returned.

Examples

```

chnFile <- system.file("extdata", "gis", "SwindaleRiverNetwork.gpkg",
                        package="dynatop", mustWork = TRUE)
chn <- terra::vect(chnFile)
chn$slope <- 1e-3
chn$class <- 1
chn$width <- 1
chn$depth <- 0.1
chn <- convert_channel(chn,
                       property_names=c(uid = "identifier",
                                         startNode = "startNode",
                                         endNode = "endNode",
                                         slope = "slope",
                                         class = "class",
                                         width="width",
                                         depth="depth"))

tchn <- trim_channel(chn,chn$uid[1])

```

trim_channel

17

```
nrow(chn)  
nrow(tchn)
```

Index

`check_channel`, [2](#)
`check_water_bodies`, [3](#)
`convert_channel`, [4](#)

`daily_evap_profile`, [5](#)
`dynatop`, [7](#)
`dynatopGIS`, [9](#)

`evap_est (daily_evap_profile)`, [5](#)

`file_functions`, [14](#)
`fill_na`, [14](#)

`read_geo (file_functions)`, [14](#)
`read_obs (file_functions)`, [14](#)
`read_param (file_functions)`, [14](#)

`simplify_channel`, [15](#)

`trim_channel`, [16](#)

`write_geo (file_functions)`, [14](#)
`write_param (file_functions)`, [14](#)