Package ‘telemac’

February 19, 2021

Type Package

Title R Interface to the TELEMAC Model Suite

Description An R interface to the TELEMAC suite for modelling of free surface flow. This includes methods for model initialisation, simulation, and visualisation. So far only the TELEMAC-2D module for 2-dimensional hydrodynamic modelling is implemented.

Version 0.1.0

URL https://github.com/tpilz/telemac

BugReports https://github.com/tpilz/telemac/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports dplyr (>= 1.0.0), fs, gstat, lubridate, magrittr, purrr, raster, Rcpp, rlang, sf, sp, stringr, sys, tidyr (>= 1.0.0), tidyselect

Suggests knitr, rgdal, rmarkdown, RTriangle, stars, terrainmeshr, tidyverse

RoxygenNote 7.1.1

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author Tobias Pilz [aut, cre] (<https://orcid.org/0000-0002-5641-3918>), Potsdam Institute for Climate Impact Research (PIK) [cph, fnd]

Maintainer Tobias Pilz <topilz@pik-potsdam.de>

Repository CRAN

Date/Publication 2021-02-19 10:10:05 UTC
R topics documented:

cas ......................................................... 2
cli ......................................................... 4
geo .......................................................... 5
interpol ..................................................... 7
lines.t2d_tin .............................................. 9
line_spacing ............................................... 9
optionals .................................................. 12
plot.t2d_geo ............................................. 13
plot.t2d_res .............................................. 14
plot.t2d_tin .............................................. 14
read_cas .................................................. 15
read_cli ................................................... 15
read_geo ................................................... 16
read_results ............................................. 17
read_slf_header ......................................... 17
read_slf_variable ....................................... 18
results .................................................... 19
simulate_t2d ............................................. 20
t2d .......................................................... 21
tin .......................................................... 22
tin2grid .................................................... 25
write_cas .................................................. 27
write_cli ................................................... 28
write_geo ................................................... 29
write_opt ................................................... 29
write_results ............................................. 30
write_slf_header ......................................... 31
write_slf_variable ....................................... 31
write_t2d .................................................. 32

Index 33

| cas | Steering file (*.cas) |

Description

Initialise a steering file for use within TELEMAC.

Usage

```r
cas(x, fname, ...)
```

## Default S3 method:
cas(x = NULL, fname = NULL, ...)

## S3 method for class 'list'
cas(x, fname = NULL, ...)

## S3 method for class 't2d_cas'
cas(x, fname = NULL, data = NULL, ...)

## S3 method for class 't2d_cas'
print(x, ..., n = 10)

### Arguments

- **x**: Either: NULL (default), in which case a simple template will be generated; a character string providing the name of an existing steering file; a list with named elements being the steering parameters and their values; an object of class t2d_cas.
- **fname**: character string providing the name of the steering file that is to be generated (can also be used to replace an existing entry).
- **...**: Arguments passed to or from other methods.
- **data**: list that can be given to update an existing object x.
- **n**: Maximum number of steering parameters to print.

### Value

An object of class t2d_cas consisting of a list with steering parameters, and an attribute file pointing to a specific steering file.

### Examples

```r
# template steering parameters
cas_tpl <- cas()

# investigate object
cas_tpl
str(cas_tpl)
class(cas_tpl) # inherits from list

cas_subset <- cas_tpl[1:5]
cas_subset

cas_updated <- cas(cas_tpl, fname = "test.cas")
cas_updated
```
cli

Boundary (*.cli) object

Description

Initialise a boundary (*.cli) object for use within TELEMAC.

Usage

cli(x, fname, ...)

## S3 method for class 'character'
cli(x, fname = NULL, ...)

## S3 method for class 'numeric'
cli(x, fname = NULL, ...)

## S3 method for class 't2d_geo'
cli(x, fname = NULL, ...)

## S3 method for class 'data.frame'
cli(x, fname = NULL, ...)

## S3 method for class 't2d_cli'
cli(x, fname = NULL, df = NULL, ...)

## S3 method for class 't2d_cli'
print(x, ..., n = 10)

Arguments

x Either: A character string providing the name of an existing boundary file; a numeric vector of ipobo values (see geo) resulting in template boundary values (closed boundary with zero prescribed depths / velocities); an object of class t2d_geo providing ipobo values converted into template boundary values; a data.frame with required columns (see 'Value') giving the boundary parameters; an object of class t2d_cli.

fname character string providing the name of the boundary file that is to be generated (can also be used to replace an existing entry).

... Arguments passed to or from other methods.

df data.frame that can be given to update an existing object x.

n Maximum number of boundary points to print.
Value

An object of class t2d_cli consisting of an attribute file pointing to a specific boundary file and a data.frame where each row represents a point along the mesh boundary. Columns refer to:

- **lihbor** Depth conditions
- **liubor** Velocity conditions in u direction
- **livbor** Velocity conditions in v direction
- **hbor** Prescribed depth if lihbor = 5
- **ubor** Prescribed velocity if liubor = 6
- **vbor** Prescribed velocity if livbor = 6
- **aubor** Friction coefficient at boundary if liubor or livbor = 2
- **litbor** Tracer conditions
- **tbor** Prescribed value of tracer if litbor = 5
- **atbor** Coefficient of flow relation
- **btbor** Coefficient of flow relation
- **n** Global number of boundary point
- **k** Point number of boundary point numbering; can also represent a node colour

See TELEMAC-2D user manual for more information.

---

**geo**

Mesh / geometry object

Description

Initialise a mesh (geometry) object for use within TELEMAC.

Usage

```r
geo(x, fname, ...) 
```

## S3 method for class 'character'
```r
geo(x, fname = NULL, ...) 
```

## S3 method for class 'list'
```r
geo(x, fname = NULL, ...) 
```

## S3 method for class 't2d_geo'
```r
geo(x, fname = NULL, data = NULL, ...) 
```

## S3 method for class 't2d_tin'
```r
geo(x, fname = NULL, ..., dem, title = "") 
```

## S3 method for class 't2d_geo'
```r
print(x, ...) 
```
Arguments

x
Either: A character string providing the name of an existing geometry file; a
list with all required elements (see 'Value'); an object of class t2d_geo; an
object of class t2d_tin (requires input dem).

fname
character, name for the associated geometry file (can also be used to replace
an existing entry).

Arguments passed to or from other methods: read_geo if x is a file name or
interpol if x is a t2d_tin object.

data
list with all required elements (see Value) that can be given to update an ex-
isting object x.

dem
If x is of type t2d_tin: a single DEM or a named list with element elevation
and further private variables from which the values will be interpolated to the
TIN mesh points. Each element will be forwarded to interpol, argument src
(see descriptions therein for supported types). Private variables must be a list
with elements values (the object passed to interpol), pars_interp (variable-
specific parameter list passed to interpol with priority over ...), and unit
(the unit of the variable). Parameters for interpolation include, e.g., the number
of neighbours n, etc.

title
character, mesh title (not necessary, max. 72 characters).

Value

An object of class t2d_geo consisting of an attribute file pointing to a specific geometry file and
a list with the following elements:

header General mesh information including title, precision (of numbers in the slf file in bytes),
the numbers of mesh elements and points, and the number of points per mesh (3 in case of
triangles which is the only supported value for now).

tin A mesh object of class t2d_tin, see tin.

elevation Values of mesh point elevations.

privars A named list of additional, in TELEMAC termed 'private' variables with elements values
and unit.

Note

Also note the associated plot.t2d_geo method.

Examples

## Not run:
library(raster)
library(sf)
library(tidyverse)

# get a tin
bnd <- st_read(system.file("dem/boundary_lagos.gpkg", package = "telemac"))
tin_obj <- tin(list(boundary = bnd), s = 90, a = 100*2, q = 30)
# load raster
dem_rast <- raster(system.file("dem/dem_merit_lagos.tif", package = "telemac"))

# create a geo object (interpolates raster to mesh points)
geo_obj <- geo(tin_obj, dem = dem_rast)
str(geo_obj)

# adjust file name
geo_obj <- geo(geo_obj, fname = "geo.slf")
geo_obj

# plot: mesh elevations interpolated to grid with resolution s
plot(geo_obj, s = 30)

# add additional private variable (in this case Curve Numbers)
dem_priv <- list(
  elevation = dem_rast,
  cn = list(values = as.data.frame(dem_rast, xy = TRUE, na.rm = TRUE) %>%
    select(x, y) %>%
    mutate(z = case_when(
      y > 740000 ~ 95,
      x > 534000 ~ 90,
      TRUE ~ 80
    )),
    unit = "-",
    pars_interp = list(n = 1)
)

geo_priv <- geo(tin_obj, dem = dem_priv)

geo_priv
str(geo_priv)
plot(geo_priv, s = 30, v = "cn")

## End(Not run)

---

**interpol**

Interpolate from source to target locations

**Description**

Interpolates values from source to target location using inverse distance weighting (IDW) of nearest neighbours.
Usage

```r
interpol(trg, src, ...)
```

```r
## S3 method for class 'SpatialPoints'
interpol(trg, src, ..., z, n = 5, output = "sp")
```

```r
## S3 method for class 'matrix'
interpol(trg, src, ..., n = 5, output = "numeric")
```

```r
## S3 method for class 'data.frame'
interpol(trg, src, ..., n = 5, output = "data.frame")
```

```r
## S3 method for class 't2d_tin'
interpol(trg, src, ..., n = 5, output = "numeric")
```

Arguments

```r
trg
```

The target locations. Either: a matrix or data.frame with x and y coordinates; an object of class SpatialPoints; an object of class t2d_tin.

```r
src
```

The source locations. Either: a matrix with x and y coordinates (argument z as to be provided); a data.frame with x and y coordinates and the variable of interest ("z"); an object of class SpatialPointsDataFrame, raster, or stars (only the first attribute is taken).

```r
...
```

Further arguments passed to idw.

```r
z
```

If src is a matrix: numeric vector of values at src locations to be interpolated to trg locations.

```r
n
```

The number of nearest neighbours used for interpolation (default is 5).

```r
output
```

The type of output: numeric, sp, or data.frame (see below).

Details

Function calls idw. You can pass further arguments to that function, e.g. idp to influence the distance-based weighting of neighbours (default is 2).

Value

```
output = "numeric": a vector of values interpolated to trg locations.
output = "data.frame": a data.frame with x and y coordinates of trg location and interpolated values ("z").
output = "sp": an object of class SpatialPointsDataFrame with the interpolated values at trg locations.
```
Add TELEMAC mesh as lines to a plot

Description

Adds the mesh (triangles) of an object of class t2d_tin as lines to an existing plot.

Usage

## S3 method for class 't2d_tin'
lines(x, ...)

Arguments

x

An object of class t2d_tin.

...

Arguments passed to plot.

Value

No return value, called for side effects (plot).

line_spacing

Adjust line vertex spacing

Description

Function harmonises the lengths of the segments of lines, i.e. the spacing of vertices.

Usage

line_spacing(x, s, output = c("df", "sp"), ...)

## S3 method for class 'data.frame'
line_spacing(
  x,
  s,
  output = c("df", "sp"),
  ...,
  col_x = "x",
  col_y = "y",
  col_line = "line"
)

## S3 method for class 'numeric'
line_spacing(x, s, output = c("df", "sp"), ..., y, line = NULL)
## S3 method for class 'matrix'
line_spacing(x, s, output = c("df", "sp"), ..., line = NULL)

## S3 method for class 'SpatialLines'
line_spacing(x, s, output = "sp", ...)

## S3 method for class 'SpatialLinesDataFrame'
line_spacing(x, s, output = "sp", ...)

### Arguments

**x**
Either: a data.frame with vertex coordinates and line identifier; a numeric vector with the x coordinates of line vertices; a matrix with two columns, the x and y coordinates of line vertices; an object of class SpatialLines*.  

**s**
numeric value giving the target spacing of line vertices (i.e. line segment lengths) in units of the input coordinates.

**output**
Return either: "df", a data.frame (default) or "sp", an object of class SpatialLines.

**...**
Arguments passed to or from other methods.

**col_x**
If x is a data.frame: column with the x coordinates.

**col_y**
If x is a data.frame: column with the y coordinates.

**col_line**
If x is a data.frame: column with the line identifier.

**y**
The y coordinates of line vertices (if x is a numeric vector).

**line**
numeric vector of identifiers to distinguish individual lines via index (or row) in x (only needed if x is a vector or matrix and more than one line is given).

### Value

If output == "df": a data.frame (or tibble) with elements x, y, and line (or col_x, col_y, col_line) defining the harmonised line(s).

If output == "sp": a SpatialLines object of the harmonised line(s).

### Note

If x is a data.frame, all input arguments referring to columns of x support quasiquotation, i.e. they can be specified by their names, quoted or unquoted, or as column position.

### Examples

```r
# one line given as numeric vectors
x = c(1,1.5,2.5,3,3.2,5, 5.8, 6.5, 7, 6.7, 6, 5.5, 4.8, 4.3, 4, 4, 4.3)
y = c(1.8,2,2.8,3.3,3.4, 4.2, 3.8, 3, 2.7, 2.5, 2.5, 2.2, 2.5, 2.8, 4, 3.2, 3.4)
plot(x, y, pch = 16, asp = 1)
lines(x, y)
ldf_harm <- line_spacing(x = x, y = y, s = 1)
points(ldf_harm$x, ldf_harm$y, pch = 16, col = "red")
lines(ldf_harm$x, ldf_harm$y, col = "red")
```
# two lines in a matrix
x2 = c(10, 12, 12.2, 11.5, 11.4, 12.5, 13.5, 14)
y2 = c(4, 4.2, 3.5, 3.3, 2.5, 2.2, 2.5, 3)
mat <- rbind(cbind(x, y), cbind(x2, y2))
line <- c(rep(1, length(x)), rep(2, length(x2)))
plot(x, y, pch = 16, xlim = c(min(mat[,1]), max(mat[,1])),
ylim = c(min(mat[,2]), max(mat[,2])), asp = 1)
points(x2, y2, pch = 16)
lines(x, y)
lines(x2, y2)
ldf_harm <- line_spacing(x = mat, line = line, s = 1)
lh1 <- ldf_harm[ldf_harm$line == 1,]
lh2 <- ldf_harm[ldf_harm$line == 2,]
points(lh1$x, lh1$y, pch = 16, col = "red")
lines(lh1$x, lh1$y, col = "red")
points(lh2$x, lh2$y, pch = 16, col = "red")
lines(lh2$x, lh2$y, col = "red")

# data.frame
library(dplyr)
line1 <- data.frame(xcoord = x, ycoord = y)
line2 <- data.frame(xcoord = x2, ycoord = y2)
ldf <- bind_rows(line1, line2, .id = "id")
plot(line1, pch = 16, xlim = c(min(ldf$x), max(ldf$x)),
ylim = c(min(ldf$y), max(ldf$y)), asp = 1)
points(line2, pch = 16)
lines(line1)
lines(line2)
ldf_harm <- line_spacing(ldf, s = 1, col_x = xcoord, col_y = ycoord, col_line = id)
line1_harm <- ldf_harm %>%
filter(id == 1) %>%
dplyr::select(xcoord, ycoord)
line2_harm <- ldf_harm %>%
filter(id == 2) %>%
dplyr::select(xcoord, ycoord)
points(line1_harm, pch = 16, col = "red")
points(line2_harm, pch = 16, col = "red")
lines(line1_harm, col = "red")
lines(line2_harm, col = "red")

# SpatialLines object
library(sp)
sl <- SpatialLines(list(Lines(Line(line1), 1), Lines(Line(line2), 2)))
plot(sl, asp = 1)
sl_harm <- line_spacing(sl, s = 1, output = "sp")
lines(sl_harm, col = "red")
Description

Initialise optional input files for TELEMAC-2D.

Usage

optionals(x, fname, ...)

## S3 method for class 'character'
optionals(x, fname, ...)

## S3 method for class 'list'
optionals(x, fname, ...)

## S3 method for class 't2d_opt'
optionals(x, fname, ..., vals)

## S3 method for class 't2d_opt'
print(x, ..., n = 10)

## S3 method for class 't2d_opt_LINES'
print(x, ..., n = 10)

Arguments

x  Either: a character vector where each element represents a line of content for the optional input file; a list with multiple character vectors to provide the contents for multiple optional files; an object of type t2d_opt to add further optional file(s).

fname  character, file name(s).

...  Arguments passed to or from other methods.

vals  If x is a t2d_opt object: a character vector or a list with the values for the additional optional input file(s).

n  Maximum number of file lines to print.

Value

An object of type t2d_opt consisting of a data.frame with elements file, file name(s) of the optional input file(s), and value, an object of type t2d_opt_LINES that is essentially a list of character vectors where each element represents the contents of an optional input file.
Note

When providing optional input files to a t2d setup the user still needs to add the respective keywords to the steering (cas) file, otherwise the optional input will be ignored! The reason is that many optional input files require additional settings (keywords in the cas file) that cannot be foreseen.
So far only text-based optional input files are supported (e.g. SECTIONS INPUT FILE or FORMATTED DATA FILE) but no binary files.

Examples

```r
# t2d_opt object
opt_obj <- optionals(c("# test file", "col1 col2", "1 2", "3 4"), fname = "optional.txt")
opt_obj
str(opt_obj)

# values (lines of the file) as t2d_opt_LINES object
opt_obj$value
str(opt_obj$value)

# multiple optional files
opt_obj <- optionals(list(c("# test file", "col1 col2", "1 2", "3 4"),
                          c("# test file 2", "col1 col2", "5 6", "7 8")),
                          fname = c("optional1.txt", "optional2.txt"))
print(opt_obj, n = 4)

# change individual values via list methods
opt_obj$value[[1]][3] <- c("10 20")
opt_obj
```

plot.t2d_geo

Plot TELEMAC geometry

Description

Plots selected objects of class t2d_geo by interpolating the underlying mesh to a regular grid and using plot of package raster.

Usage

```r
## S3 method for class 't2d_geo'
plot(x, s, v = "elevation", ...)
```

Arguments

- **x**: An object of class t2d_geo.
- **s**: numeric value defining the resolution for plotting (the mesh has to be interpolated to a grid for that).
- **v**: character, name of the variable that is to be plotted (only one at a time; default is elevation).
- **...**: Arguments passed to plot.
**Value**

No return value, called for side effects (plot).

---

**plot.t2d_res**  
*Plot TELEMAC results*

**Description**

Plots selected variable and timestep of objects of class t2d_res by interpolating the underlying mesh to a regular grid and using `plot` of package `raster`.

**Usage**

```r
## S3 method for class 't2d_res'
plot(x, s, v = NULL, t = 0, ...)
```

**Arguments**

- `x`: An object of class t2d_res.
- `s`: numeric value defining the resolution for plotting (the mesh has to be interpolated to a grid for that).
- `v`: character, name of the variable that is to be plotted (only one at a time; default is to take the first variable that can be found).
- `t`: integer, timestep that is to be plotted (only one at a time; default is to plot the first timestep).
- `...`: Arguments passed to `plot`.

**Value**

No return value, called for side effects (plot).

---

**plot.t2d_tin**  
*Plot TELEMAC mesh*

**Description**

Plots the mesh (triangles) of an object of class t2d_tin.

**Usage**

```r
## S3 method for class 't2d_tin'
plot(x, ...)
```
read_cas

Arguments

- x: An object of class t2d_tin.
- ...: Arguments passed to plot.

Value

No return value, called for side effects (plot).

---

Description

Reads the steering file of a TELEMAC project.

Usage

read_cas(fname)

Arguments

- fname: character File name of the steering file to be read.

Value

A list with steering parameters and their values.

See Also

To obtain a t2d_cas object use function cas.

---

read_cli

Read boundary (*.cli) file

---

Description

Reads the boundary file of a TELEMAC project.

Usage

read_cli(fname)

Arguments

- fname: character File name of the boundary file to be read.
Value
A data.frame where each row represents a point along the mesh boundary. Columns refer to:

- **lihbor** Depth conditions
- **liubor** Velocity conditions in u direction
- **livbor** Velocity conditions in v direction
- **hbor** Prescribed depth if lihbor = 5
- **ubor** Prescribed velocity if liubor = 6
- **vbor** Prescribed velocity if livbor = 6
- **aubor** Friction coefficient at boundary if liubor or livbor = 2
- **litbor** Tracer conditions
- **tbor** Prescribed value of tracer if litbor = 5
- **atbor** Coefficient of flow relation
- **btbor** Coefficient of flow relation
- **n** Global number of boundary point
- **k** Point number of boundary point numbering; can also represent a node colour

See TELEMAC-2D user manual for more information.

---

**read_geo**  
**Read geometry file (*.slf)**

**Description**
Reads the geometry / mesh information of a TELEMAC project from a SELAFIN file

**Usage**
```r
read_geo(fname, privar = TRUE)
```

**Arguments**
- **fname** character: File name of the geometry file to be read.
- **privar** logical, shall variables in addition to bottom elevation (in TELEMAC denoted as private variables) be imported if available? Default: TRUE.

**Details**
At least variable `BOTTOM` (bottom elevations of the mesh) is expected in the geometry file.

**Value**
A list with header information (see output of `read_slf_header`); elevation, the mesh point elevations; `privars`, named list of additional 'private' variables.
**read_results**

Read results from *.slf*

**Description**

Reads the results of a TELEMAC simulation from a SELAFIN (*.slf) file.

**Usage**

```r
read_results(fname, vars = "all", times = NULL, return_datetime = FALSE)
```

**Arguments**

- `fname` character
  File name of the results file to be read.
- `vars` character
  Selection of variables to be read from the file: either "all" (default) reading all variables, "none" giving only the header, a character vector of variable names, or a numeric vector of positions.
- `times` integer
  Select the timesteps to be read. Passed to `read_slf_header`.
- `return_datetime` logical
  Return timesteps as datetime (POSIXct) object?

**Value**

A list with header (see output of `read_slf_header`) and values, which is a tidy data.frame where each line represents the value for a certain mesh point (with coordinates x and y) at a certain simulation timestep (note that this might be difficult to interpret if you used variable timestep lengths) for a specific variable.

---

**read_slf_header**

Read slf header

**Description**

Reads the header of a SELAFIN file (*.slf).

**Usage**

```r
read_slf_header(fname)
```

**Arguments**

- `fname` character
  Name of a SELAFIN file.
Value

A list with the following elements:

- **title** Title of the file
- **precision** Precision of numbers (bytes)
- **nv** Number of variables
- **varnames** Names of variables
- **varunits** Units of variables
- **date** Starting date-time (POSIXct object)
- **ntimes** Number of timesteps
- **nelem** Number of mesh elements (triangles)
- **npoi** Number of mesh points
- **ndp** Number of points per element (3 in case of triangles)
- **ikle** A nelem x ndp integer matrix of point indices (referring to x and y) defining the mesh elements
- **ipobo** An integer vector of length npoi defining the mesh boundaries (inner boundaries are zero, outer boundaries numbered)
- **x** A vector of length npoi giving the x coordinates of the mesh points
- **y** A vector of length npoi giving the y coordinates of the mesh points

**seek_head** Position in the SELAFIN file where the header ends (required by function `read_slf_variable`)

---

### Description

Reads the variables stored in a SELAFIN file (*.slf).

### Usage

```r
read_slf_variable(fname, seek_start, vars, nv, fsize, npoin, times = NULL)
```

### Arguments

- **fname** character, name of a SELAFIN file.
- **seek_start** integer, starting position to read data from the binary file; can be inferred with function `read_slf_header`.
- **vars** numeric, vector giving the indices of the variables that shall be read (can be inferred from header information, `read_slf_header`).
- **nv** integer giving the total number of variables in the file.
- **fsize** integer, precision of the values (number of bytes).
- **npoin** integer, number of mesh points.
- **times** integer vector, the timesteps to be read. If any specified timestep cannot be found in the file an error is raised. Default (NULL): all.
## Value

A list with the following elements:

- **time** Timesteps in seconds relative to date (element in output of `read_slf_header`)
- **values** An array of dimensions \((\text{npoint} \times \text{nv} \times \text{length(time)})\) with the values of the variables for each meshpoint and timestep

---

## Description

Initialise a results object to handle the results of TELEMAC-2D runs.

## Usage

```r
results(x, ...)  
## Default S3 method:  
results(x = NULL, fname = "results.slf", ...)  
## S3 method for class 'character'  
results(x, fname = NULL, log = NULL, ...)  
## S3 method for class 't2d_res'  
results(x, fname = NULL, log = NULL, ...)  
## S3 method for class 't2d'  
results(x, ...)  
## S3 method for class 't2d_res'  
print(x, ..., n = 10)
```

### Arguments

- **x** Either: NULL (default), in which case a simple template without data will be generated; a character string providing the name of an existing results file; an object of class `t2d_res` (modify attributes); an object of class `t2d` (read data after model run).
- **...** Arguments passed to or from other methods, e.g. to `read_results` such as arguments `vars`, `times`, or `return_datetime`.
- **fname** character, name for the associated results file (can also be used to replace an existing entry).
- **log** File name of a log file from a simulation run, if it exists.
- **n** Maximum number of steering parameters to print.
simulate_t2d

Value
An object of class t2d_res consisting of an attribute file pointing to a specific results file, an attribute log pointing to the log of a simulation run, and a list with elements:

header General information including title, precision (of numbers in the slf file in bytes), the numbers of mesh elements and points, the number of points per mesh (3 in case of triangles which is the only supported value for now), the variable names and units, and the simulation start date.

tin An object of class t2d_tin representing the underlying mesh.

log The log messages (a character vector).

values A data.frame where each line represents the value for a certain mesh point (with coordinates x and y) at a certain simulation timestep (note that this might be difficult to interpret if you used variable timestep lengths) for a specific variable.

Note
Also note the associated plot.t2d_res method.

simulate_t2d

TELEMAC-2D model run

Description
Conduct a TELEMAC-2D model run using the model’s system command.

Usage
simulate_t2d(x, log = "run.log", res = NULL, vars = "all", exec)

Arguments
x An object of class t2d.

log character, a filename for logging of runtime messages to be created in the project directory (x$wdir). Default: run.log. Set to NULL if no logfile shall be created (in that case the runlog will not be checked for errors).

res character, the results file (must be of type *_.slf). If NULL (default), the information from the existing steering file will be used or (if the file does not yet exist) will be extracted from x$res.

vars Selection of variables to be imported after the simulation: either "all" (default) reading all variables, "none" giving only the header, a character vector of variable names, or a numeric vector of positions. Importing all variables may require large storage capacities (depending on the number of variables, mesh points, and specified output intervals).

exec character, the TELEMAC-2D executable if not already specified in x.
**Value**

An object of class `t2d` with new or updated element `res` (an object of class `t2d_res`).

**Note**

In case the project directory or any of the mandatory input files do not yet exist, this function will first call `write_t2d` and then run the model.

TELEMAC-2D runs are often rather long. During simulation time this command will, by default, write nothing to console but store the output in the specified log file.

In case of large projects with expected long model runtimes it might make more sense to run TELEMAC-2D directly instead of using this function.

---

**t2d**

*Initialise a TELEMAC-2D project*

---

**Description**

Initialises a project for 2-dimensional hydrodynamic modelling with TELEMAC-2D.

**Usage**

```r
?t2d
```

```r
## S3 method for class 't2d'
print(x, ...)
```

**Arguments**

- **title**
  - character string giving the title of the project (argument TITLE in the steering file).
- **wdir**
  - character string, the project directory were inputs and outputs will be written to. TELEMAC-2D’s input filenames must be relative to this directory! Default: current working directory.
- **cas**
  - Passed to `cas` (preferably a `t2d_cas` object).
- **geo**
  - Passed to `geo` (preferably a `t2d_geo` object).
- **cli**
  - Passed to `cli` (preferably a `t2d_cli` object).
- **res**
  - Passed to `results`. Can be the name of an existing results file or left empty at the beginning in which case an empty template object will be generated.
- **opt**
  - Object of class `t2d_opt` (if any; default: NULL).
- **exec**
  - character string specifying the TELEMAC-2D executable (system command). Default: NULL. Only required for `simulate_t2d` to work.
- **x**
  - An object of class `t2d`.
- **...**
  - Optional arguments passed to print methods.
Details

First, make sure TELEMAC-2D is installed and works!

Value

An object of class t2d, that is a list with elements title, wdir, exec, cas, geo, cli, res, and opt. The latter five are objects of type t2d_*.

Examples

```r
library(sf)
library(raster)

# template setup with example data
bnd <- st_read(system.file("dem/boundary_lagos.gpkg", package = "telemac"))
dem_rast <- raster(system.file("dem/dem_merit_lagos.tif", package = "telemac"))
tin_obj <- tin(list(boundary = bnd), s = 90, a = 100^2, q = 30)
geo_obj <- geo(tin_obj, dem = dem_rast)
cli_obj <- cli(geo_obj)
cas_obj <- cas()

# TELEMAC-2D setup
t2d_obj <- t2d("Test setup", "path/to/wdir",
               cas = cas_obj, geo = geo_obj, cli = cli_obj)
t2d_obj
```

---

**tin**

*TIN object*

Description

Initialise a TIN mesh object for use within TELEMAC.

Usage

```r
tin(x, ...)

## S3 method for class 'character'
tin(x, ...)

## S3 method for class 'matrix'
tin(x, ..., ike, ipobo)

## S3 method for class 'list'
tin(x, ..., s, s_brk, a, q = 30)

## S3 method for class 't2d_tin'
print(x, ...)
```
Arguments

\textbf{x} \hspace{1cm} \text{Either: a character string providing the name of a SELAFIN file of which the mesh will be extracted; a matrix of mesh points with 2 columns containing the x and y coordinates (arguments \texttt{ikle} and \texttt{ipobo} are required); a list with boundary and breakline definitions (see Details).}

\ldots \hspace{1cm} \text{Arguments passed to or from other methods. If x is a list, further arguments passed to \texttt{triangulate}.}

\textbf{ikle} \hspace{1cm} \text{If x is a matrix of points: A matrix with 3 columns of indices referring to rows in x; each row represents a mesh element (triangle). In TELEMAC termed \texttt{IKLE}.}

\textbf{ipobo} \hspace{1cm} \text{If x is a matrix of points: A vector of indices referring to rows in x, each marking a boundary point. In TELEMAC termed \texttt{IPOBO}.}

\textbf{s} \hspace{1cm} \text{numeric, if x is a list: OPTIONAL value giving the resolution of vertices along the boundary line for triangulation. If not given, the points are used as they are supplied, otherwise \texttt{line\_spacing} is called to ensure equal spacing of vertices with the given segment lengths.}

\textbf{s\_brk} \hspace{1cm} \text{As s but for breaklines.}

\textbf{a} \hspace{1cm} \text{numeric, maximum triangle area; passed to \texttt{triangulate}. Default: squared spacing of points (either given as s or inferred from x$boundary).}

\textbf{q} \hspace{1cm} \text{numeric, minimum triangle angle; passed to \texttt{triangulate}. Default: 30 degrees.}

Details

If \texttt{x is a list} this function creates a Triangulated Irregular Network (TIN) using function \texttt{triangulate}. The following list elements are required to perform the triangulation:

\textbf{boundary} \hspace{1cm} \text{A matrix, data.frame, SpatialLines* or sf object with two columns, each row defining a point along the outer catchment boundary. Points are connected one-by-one to a line starting with the first point, i.e. make sure points are in the right order! The first and last point will be connected to close the boundary.}

\textbf{breaklines} \hspace{1cm} \text{OPTIONAL, a matrix, data.frame, SpatialLines* or sf object with three columns \texttt{x} and \texttt{y}, the \texttt{x} and \texttt{y} coordinates of vertices along the breaklines, and \texttt{line}, an identifier to identify individual breaklines.}

Value

An object of class \texttt{t2d_tin}, which is a list with the following elements:

\textbf{points} \hspace{1cm} \text{A matrix with the \texttt{x} and \texttt{y} coordinates (as columns) of mesh points.}

\textbf{triangles} \hspace{1cm} \text{A matrix with 3 columns of indices referring to rows in \texttt{points}; each row represents a mesh element (triangle).}

\textbf{edges} \hspace{1cm} \text{A matrix with 2 columns of indices referring to rows in \texttt{points}, the node points; each row represents an edge / segment of a triangle.}

\textbf{boundaries} \hspace{1cm} \text{A vector of indices referring to rows in \texttt{points}, each marking a point of the outer catchment boundary.}

\textbf{breaklines} \hspace{1cm} \text{A matrix with 2 columns of indices referring to rows in \texttt{points}, the vertices of the breaklines (used for mesh refinement during triangulation).}
Note

Duplicated mesh points are silently removed.

Make sure breaklines do not intersect as this is not supported by the Triangle algorithm. A possible workaround to split intersecting breaklines in R using [sf](https://r-spatial.github.io/sf) is shown in the examples.

If you want to construct a `t2d_tin` object and get the error `Boundary points do not comply with requirements: [...]` the reason might be that breaklines are too close to the boundary causing that points of the breaklines are used as boundary points which eventually results in a discontinuous outer boundary. Try to increase the distance of breaklines to the catchment boundary.

Examples

```r
### BASIC FUNCTIONALITY ###
library(sf)

# load boundary as sf linestring
bnd <- st_read(system.file("dem/boundary_lagos.gpkg", package = "telemac"))

# create t2d_tin object
tin_obj <- tin(list(boundary = bnd), s = 90, a = 100^2, q = 30)

# inspection
tin_obj
str(tin_obj)
plot(tin_obj, pch = ".")

### DEALING WITH INTERSECTING BREAKLINES ###
library(sf)
library(tidyverse)

# example boundary and with intersecting breaklines
test_bnd <- st_linestring(
    matrix(c(seq(0,100,5), rep(100,21), seq(100,0,-5), rep(0,21), rep(0,21), seq(0,100,5), rep(100,21), seq(100,0,-5)),
        ncol = 2)
  ) %>% st_sfc()

test_brk <- list(
    st_linestring(matrix(c(seq(0,100,5), rep(50,21)), ncol = 2)),
    st_linestring(matrix(c(rep(50,21), seq(0,100,5)), ncol = 2)),
    st_linestring(matrix(c(seq(30,60,5), rep(60,11), rep(20,7), seq(20,70,5)), ncol = 2))) %>% st_sfc()

# get intersection points and define buffer of 2 around these points
pt_inters <- c(test_bnd, test_brk) %>%
    st_intersection() %>%
    st_collection_extract(type = "POINT") %>%
    st_buffer(2)

plot(test_bnd)
plot(test_brk, add = TRUE)
```
plot(pt_inters, add = TRUE)

# split breaklines
test_brk_unique <- st_difference(st_union(test_brk), st_union(pt_inters))
plot(test_bnd)
plot(test_brk_unique, add = TRUE)

# create mesh
tin_obj <- tin(list(boundary = test_bnd, breaklines = test_brk_unique),
                s = 2, s_brk = 2, a = 4, q = 30)
plot(tin_obj, pch = ".")

---

### tin2grid

Interpolate TIN-based mesh values to grid

#### Description

Linearly interpolates the values of a mesh based on irregular triangles to a regular grid.

#### Usage

```r
tin2grid(x, s, output, ...)
```

#### S3 method for class 'data.frame'

```r
tin2grid(
  x,
  s,
  output = c("data.frame", "list", "raster"),
  ..., col_x = "x",
  col_y = "y",
  col_z = "z",
  tinmat
)
```

#### S3 method for class 't2d_geo'

```r
tin2grid(
  x,
  s,
  output = c("data.frame", "list", "raster"),
  ..., v = "elevation"
)
```

#### S3 method for class 't2d_res'

```r
tin2grid(
  x,
  s,
)```
output = c(”data.frame”, ”list”, ”raster”),
..., 
v = NULL,
t = NULL
)

Arguments

x Either: a data.frame with mesh coordinates and elevations; an object of class t2d_geo or t2d_res.
s numeric value defining the resolution of the output grid.
output character, either: ”data.frame” (default), ”list”, or ”raster” to return a raster object. If multiple variables v or timesteps t are given, a data.frame will be returned in any case.
...
Arguments passed to or from other methods.
col_x If x is a data.frame: column with the x coordinates.
col_y If x is a data.frame: column with the y coordinates.
col_z If x is a data.frame: column with the mesh values.
tinmat integer matrix of point indices (referring to x and y coordinates) defining the mesh elements (such as element triangles of a t2d_tin object).
v character, name of the variable(s) that is/are to be extracted and interpolated (default all variables that can be found). If x is of class t2d_geo elevation will be taken by default.
t integer, timestep(s) that is/are to be extracted and interpolated (default: all timesteps).

Value

If output == ”data.frame”: A data.frame with:

x or col_x X coordinates of the output grid
y or col_y Y coordinates of the output grid
z or col_z Interpolated values

variable OPTIONAL (more than one given): imported variable the current values refer to
timestep OPTIONAL (more than one given): simulation timestep the current values refer to

If output == ”list”: A list with:

x X coordinates of the output grid
y Y coordinates of the output grid
z Matrix with interpolated values, where z[i,j] points to coordinates x[i] and y[j]

If output == ”raster”: An object of class raster.
Note

If you import many variables or timesteps or the mesh is huge or s very small the resulting dataset might become excessively large (especially if output is a data.frame)!

If x is a data.frame, all input arguments referring to columns of x support quasiquote, i.e. they can be specified by their names, quoted or unquoted, or as column position.

Examples

```r
# plot model bathymetry with ggplot
library(raster)
library(sf)
library(tidyverse)
# t2d_geo object
bnd <- st_read(system.file("dem/boundary_lagos.gpkg", package = "telemac"))
tin_obj <- tin(list(boundary = bnd), s = 90, a = 100^2, q = 30)
dem_rast <- raster(system.file("dem/dem_merit_lagos.tif", package = "telemac"))
geo_obj <- geo(tin_obj, dem = dem_rast)
# interpolate to regular grid as df and plot
geo_df <- tin2grid(geo_obj, s = 90, output = "data.frame")
ggplot(geo_df, aes(x = x, y = y, fill = value)) +
  geom_raster() +
  coord_equal() +
  scale_fill_viridis_c()
```

write_cas

Write steering file (*.cas)

Description

Writes the steering file for a TELEMAC project.

Usage

```r
write_cas(x, ...)

## S3 method for class 'data.frame'
write_cas(x, fname, ...)

## S3 method for class 'list'
write_cas(x, fname, ...)

## S3 method for class 't2d_cas'
write_cas(x, ...)

## S3 method for class 't2d'
write_cas(x, ...)
```
**Arguments**

- **x**: Either: An object of class `t2d_cas`; a list with elements as in a `t2d_cas` object; a `data.frame` with columns `key` and `value` giving the steering parameters; an object of class `t2d` with element `cas`.
- ...: Arguments passed to or from other methods.
- **fname**: character, a file name (extension .cas) where the steering parameters should be written to.

**Value**

Returns input `x` invisibly.

**Note**

An existing steering file will be silently overwritten.

**Examples**

```r
## Not run:
# creates test.cas in current working directory
cas_tpl <- cas()
cas_tpl <- cas(cas_tpl, fname = "test.cas")
write_cas(cas_tpl)
## End(Not run)
```

---

**write_cli**

*Write boundary file (*.cli)*

**Description**

Writes boundary conditions for a TELEMAC simulation.

**Usage**

```r
write_cli(x)
```

### S3 method for class 't2d_cli'

```r
write_cli(x)
```

### S3 method for class 't2d'

```r
write_cli(x)
```

**Arguments**

- **x**: An object of class `t2d_cli` or `t2d`. 
Description
Writes geometry / mesh information into a SELAFIN file.

Usage

```r
write_geo(x)
```  
```
## S3 method for class 't2d_geo'
write_geo(x)
```
```
## S3 method for class 't2d'
write_geo(x)
```

Arguments

- `x` An object of class `t2d_geo` or `t2d`.

Value
Returns input `x` invisibly.

Description
Writes optional TELEMAC-2D input file(s).

Usage

```r
write_opt(x)
```  
```
## S3 method for class 't2d_opt'
write_opt(x)
```
```
## S3 method for class 't2d'
write_opt(x)
```

Arguments

x                   An object of class t2d_opt or t2d.

Value

Returns input x invisibly.

Description

Writes data into a SELAFIN (*.slf) file.

Usage

\[
\text{write_results(x)}
\]

\[
\text{## S3 method for class 't2d_res'}
\text{write_results(x)}
\]

\[
\text{## S3 method for class 't2d'}
\text{write_results(x)}
\]

Arguments

x                   An object of class t2d_res or t2d.

Value

An updated version of input x.

Note

Writes only the available values (x$values) into the file and adapts the header accordingly in case the header is referring to more variables that are not imported.
write_slf_header

Write slf header

Description

Writes the header of a SELAFIN file (*.slf).

Usage

write_slf_header(fname, header)

Arguments

fname character, name of the SELAFIN file to be created.
header A list with all header information (see output of read_slf_header).

Value

No return value, called for side effects.

Note

If a file fname already exists, it will be silently overwritten.
Make sure the datatypes in header (integer vs. double) match (compare with output of read_slf_header).

write_slf_variable

Write slf variable

Description

Writes variable values into a SELAFIN file (*.slf).

Usage

write_slf_variable(fname, data)

Arguments

fname character, name of the SELAFIN file (has to exist already and contain header information).
data A list with variable values; see output of read_slf_variable for the required elements; in addition an element 'precision' (see read_slf_header) is needed.

Value

No return value, called for side effects.
write_t2d  Write TELEMAC-2D files

Description

Writes all input files of a TELEMAC-2D project setup.

Usage

write_t2d(x)

Arguments

  x  An object of class t2d.

Value

Returns input x invisibly.

Note

This function is basically a wrapper around other write_* functions of the telemac package.
The associated file of x$res will replace entry RESULTS FILE in the steering parameters.
If x$wdir does not exist, it will be created.
If x$wdir is a relative path it will be considered relative to the current working directory.
All t2d*_ objects will be written to the associated filenames into directory wdir (element of x). In
case the associated filenames contain a relative path the files will be written relative to wdir.
Existing files will be silently overwritten!
Parameters BOUNDARY CONDITIONS FILE, GEOMETRY FILE, RESULTS FILE, and TITLE will be adapted
to the current setup.
# Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cas</td>
<td>2, 15, 21</td>
</tr>
<tr>
<td>cli</td>
<td>4, 21</td>
</tr>
<tr>
<td>geo</td>
<td>4, 5, 21</td>
</tr>
<tr>
<td>idw</td>
<td>8</td>
</tr>
<tr>
<td>interpol</td>
<td>6, 7</td>
</tr>
<tr>
<td>line_spacing</td>
<td>9</td>
</tr>
<tr>
<td>lines.t2d_tin</td>
<td>9</td>
</tr>
<tr>
<td>optionals</td>
<td>12</td>
</tr>
<tr>
<td>plot</td>
<td>9, 13–15</td>
</tr>
<tr>
<td>plot.t2d_geo</td>
<td>6, 13</td>
</tr>
<tr>
<td>plot.t2d_res</td>
<td>14, 20</td>
</tr>
<tr>
<td>plot.t2d_tin</td>
<td>14</td>
</tr>
<tr>
<td>POSIXct</td>
<td>17, 18</td>
</tr>
<tr>
<td>print.t2d(t2d)</td>
<td>21</td>
</tr>
<tr>
<td>print.t2d_cas(cas)</td>
<td>2</td>
</tr>
<tr>
<td>print.t2d_cli(cli)</td>
<td>4</td>
</tr>
<tr>
<td>print.t2d_geo(geo)</td>
<td>5</td>
</tr>
<tr>
<td>print.t2d_opt(optionals)</td>
<td>12</td>
</tr>
<tr>
<td>print.t2d_opt_LINES(optionals)</td>
<td>12</td>
</tr>
<tr>
<td>print.t2d_res(results)</td>
<td>19</td>
</tr>
<tr>
<td>print.t2d_tin(tin)</td>
<td>22</td>
</tr>
<tr>
<td>quasiquotation</td>
<td>10, 27</td>
</tr>
<tr>
<td>raster</td>
<td>8, 13, 14, 26</td>
</tr>
<tr>
<td>read_cas</td>
<td>15</td>
</tr>
<tr>
<td>read_cli</td>
<td>15</td>
</tr>
<tr>
<td>read_geo</td>
<td>6, 16</td>
</tr>
<tr>
<td>read_results</td>
<td>17, 19</td>
</tr>
<tr>
<td>read_slf_header</td>
<td>16, 17, 18, 19, 31</td>
</tr>
<tr>
<td>read_slf_variable</td>
<td>18, 19, 31</td>
</tr>
<tr>
<td>results</td>
<td>19, 21</td>
</tr>
<tr>
<td>simulate_t2d</td>
<td>20, 21</td>
</tr>
<tr>
<td>SpatialLines</td>
<td>10</td>
</tr>
<tr>
<td>SpatialPoints</td>
<td>8</td>
</tr>
<tr>
<td>SpatialPointsDataFrame</td>
<td>8</td>
</tr>
<tr>
<td>t2d</td>
<td>21</td>
</tr>
<tr>
<td>tibble</td>
<td>10</td>
</tr>
<tr>
<td>tin</td>
<td>6, 22</td>
</tr>
<tr>
<td>tin2grid</td>
<td>25</td>
</tr>
<tr>
<td>triangulate</td>
<td>23</td>
</tr>
<tr>
<td>write_cas</td>
<td>27</td>
</tr>
<tr>
<td>write_cli</td>
<td>28</td>
</tr>
<tr>
<td>write_geo</td>
<td>29</td>
</tr>
<tr>
<td>write_opt</td>
<td>29</td>
</tr>
<tr>
<td>write_results</td>
<td>30</td>
</tr>
<tr>
<td>write_slf_header</td>
<td>31</td>
</tr>
<tr>
<td>write_slf_variable</td>
<td>31</td>
</tr>
<tr>
<td>write_t2d</td>
<td>21, 32</td>
</tr>
</tbody>
</table>