Package: ncdfgeom (via r-universe)

September 3, 2024

Type Package

Title 'NetCDF' Geometry and Time Series

Version 1.2.0

Description Tools to create time series and geometry 'NetCDF' files.

URL https://code.usgs.gov/water/ncdfgeom

BugReports https://github.com/DOI-USGS/ncdfgeom/issues

Imports RNetCDF, ncmeta, sf, dplyr, methods, stars, areal

Depends R (>= 3.5)

Suggests testthat, knitr, rmarkdown, pkgdown, jsonlite, zip, ncdf4, nhdplusTools

License CC0

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.2.3

Repository https://doi-usgs.r-universe.dev

RemoteUrl https://github.com/doi-usgs/ncdfgeom

RemoteRef HEAD

RemoteSha 46f1b2bef68e47af971d110bd6dfab7c16daf19d

Contents

`calculate_area_intersection_weights` *Area Weighted Intersection (areal implementation)*

Description

Returns the fractional percent of each feature in x that is covered by each intersecting feature in y. These can be used as the weights in an area-weighted mean overlay analysis where x is the data source and area- weighted means are being generated for the target, y.

This function is a lightwieght wrapper around the functions aw_intersect aw_total and aw_weight from the areal package.

Usage

```
calculate_area_intersection_weights(x, y, normalize, allow_lonlat = FALSE)
```

Arguments

`x`	sf data.frame source features including one geometry column and one identifier column
`y`	sf data.frame target features including one geometry column and one identifier column
`normalize`	logical return normalized weights or not. See details and examples.
`allow_lonlat`	boolean If FALSE (the default) lon/lat target features are not allowed. Intersections in lon/lat are generally not valid and problematic at the international date line.

Details

Two versions of weights are available:

'normalize = FALSE', if a polygon from x is entirely within a polygon in y, w will be 1. If a polygon from x is 50 will be two rows, one for each x/y pair of features with w = 0.5 in each. Weights will sum to 1 per SOURCE polygon if the target polygons fully cover that feature. 'normalize = TRUE', weights are divided by the target polygon area such that weights sum to 1 per TARGET polygon if the target polygon is fully covered by source polygons.

Value

data.frame containing fraction of each feature in x that is covered by each feature in y.

Examples

```
b1 = sf::st_polygon(list(rbind(c(-1,-1), c(1,-1),
                               c(1,1), c(-1,1),
                               c(-1,-1))))
b2 = b1 + 2
```

```
b3 = b1 + c(-0.2, 2)
b4 = b1 + c(2.2, 0)
b = sf::st_sfc(b1, b2, b3, b4)
a1 = b1 * 0.8
a2 = a1 + c(1, 2)
a3 = a1 + c(-1, 2)
a = sf::st_sfc(a1,a2,a3)
plot(b, border = 'red')
plot(a, border = 'green', add = TRUE)

sf::st_crs(b) <- sf::st_crs(a) <- sf::st_crs(5070)

b <- sf::st_sf(b, data.frame(idb = c(1, 2, 3, 4)))
a <- sf::st_sf(a, data.frame(ida = c(1, 2, 3)))

sf::st_agr(a) <- sf::st_agr(b) <- "constant"

calculate_area_intersection_weights(a, b, normalize = FALSE)
calculate_area_intersection_weights(a, b, normalize = TRUE)
calculate_area_intersection_weights(b, a, normalize = FALSE)
calculate_area_intersection_weights(b, a, normalize = TRUE)

#a more typical arrangement of polygons

b1 = sf::st_polygon(list(rbind(c(-1,-1), c(1,-1),
                           c(1,1), c(-1,1),
                           c(-1,-1))))
b2 = b1 + 2
b3 = b1 + c(0, 2)
b4 = b1 + c(2, 0)
b = sf::st_sfc(b1, b2, b3, b4)
a1 = b1 * 0.75 + c(-.25, -.25)
a2 = a1 + 1.5
a3 = a1 + c(0, 1.5)
a4 = a1 + c(1.5, 0)
a = sf::st_sfc(a1,a2,a3, a4)
plot(b, border = 'red', lwd = 3)
plot(a, border = 'green', add = TRUE)

sf::st_crs(b) <- sf::st_crs(a) <- sf::st_crs(5070)

b <- sf::st_sf(b, data.frame(idb = c(1, 2, 3, 4)))
a <- sf::st_sf(a, data.frame(ida = c("a", "b", "c", "d")))

sf::st_agr(a) <- sf::st_agr(b) <- "constant"

# say we have data from `a` that we want sampled to `b`.
# this gives the percent of each `a` that intersects each `b`

(a_b <- calculate_area_intersection_weights(a, b, normalize = FALSE))

# note that `w` sums to 1 where `b` completely covers `a`.
```

```
dplyr::summarize(dplyr::group_by(a_b, ida), w = sum(w))

# We can apply these weights like...
dplyr::tibble(ida = unique(a_b$ida),
                  val = c(1, 2, 3, 4)) |>
  dplyr::left_join(a_b, by = "ida") |>
  dplyr::mutate(val = ifelse(is.na(w), NA, val),
                areasqkm = 1.5 ^ 2) |> # area of each polygon in `a`
  dplyr::group_by(idb) |> # group so we get one row per `b`
  # now we weight by the percent of the area from each polygon in `b` per polygon in `a`
  dplyr::summarize(new_val = sum( (val * w * areasqkm), na.rm = TRUE ) / sum(w * areasqkm))

# we can go in reverse if we had data from b that we want sampled to a

(b_a <- calculate_area_intersection_weights(b, a, normalize = FALSE))

# note that `w` sums to 1 only where `a` complete covers `b`

dplyr::summarize(dplyr::group_by(b_a, idb), w = sum(w))

# with `normalize = TRUE`, `w` will sum to 1 when the target
# completely covers the source rather than when the source completely
# covers the target.

(a_b <- calculate_area_intersection_weights(a, b, normalize = TRUE))

dplyr::summarize(dplyr::group_by(a_b, idb), w = sum(w))

(b_a <- calculate_area_intersection_weights(b, a, normalize = TRUE))

dplyr::summarize(dplyr::group_by(b_a, ida), w = sum(w))

# We can apply these weights like...
# Note that we don't need area in the normalized case
dplyr::tibble(ida = unique(a_b$ida),
                  val = c(1, 2, 3, 4)) |>
  dplyr::left_join(a_b, by = "ida") |>
  dplyr::mutate(val = ifelse(is.na(w), NA, val)) |>
  dplyr::group_by(idb) |> # group so we get one row per `b`
  # now we weight by the percent of the area from each polygon in `b` per polygon in `a`
  dplyr::summarize(new_val = sum( (val * w), na.rm = TRUE ))
```

`create_cell_geometry` *Create Cell Geometry*

Description

Creates cell geometry from vectors of X and Y positions.

Usage

```
create_cell_geometry(
  X_coords,
  Y_coords,
  prj,
  geom = NULL,
  buffer_dist = 0,
  regularize = FALSE,
  eps = 1e-10
)
```

Arguments

X_coords	numeric center positions of X axis indices
Y_coords	numeric center positions of Y axis indices
prj	character proj4 string for x and y
geom	sf data.frame with geometry that cell geometry should cover
buffer_dist	numeric a distance to buffer the cell geometry in units of geom projection
regularize	boolean if TRUE, grid spacing will be adjusted to be exactly equal. Only applies to 1-d coordinates.
eps	numeric sets tolerance for grid regularity.

Details

Intersection is performed with cell centers then geometry is constructed. A buffer may be required to fully cover geometry with cells.

Examples

```
dir <- tempdir()
ncf <- file.path(dir, "metdata.nc")

try(zip::unzip(system.file("extdata/metdata.zip", package = "ncdfgeom"), exdir = dir))

if(file.exists(ncf)) {

nc <- RNetCDF::open.nc(ncf)
ncmeta::nc_vars(nc)
variable_name <- "precipitation_amount"
cv <- ncmeta::nc_coord_var(nc, variable_name)

x <- RNetCDF::var.get.nc(nc, cv$X, unpack = TRUE)
y <- RNetCDF::var.get.nc(nc, cv$Y, unpack = TRUE)

prj <- ncmeta::nc_gm_to_prj(ncmeta::nc_grid_mapping_atts(nc))

geom <- sf::read_sf(system.file("shape/nc.shp", package = "sf"))
geom <- sf::st_transform(geom, 5070)
```

```
cell_geometry <- create_cell_geometry(x, y, prj, geom, 0)

plot(sf::st_geometry(cell_geometry), lwd = 0.25)
plot(sf::st_transform(sf::st_geometry(geom), prj), add = TRUE)

}
```

`read_attribute_data` *Read attribute dataframe from NetCDF-DSG file*

Description

Gets attribute data from a NetCDF-DSG file and returns it in a `data.frame`. This function is intended as a convenience to be used within workflows where the netCDF file is already open and well understood.

Usage

```
read_attribute_data(nc, instance_dim)
```

Arguments

`nc`	A NetCDF path or urlto be opened.
`instance_dim`	The NetCDF instance/station dimension.

Examples

```
hucPolygons <- sf::read_sf(system.file('extdata','example_huc_eta.json', package = 'ncdfgeom'))
hucPolygons_nc <- ncdfgeom::write_geometry(tempfile(), hucPolygons)

read_attribute_data(hucPolygons_nc, "instance")
```

`read_geometry` *Read NetCDF-CF spatial geometries*

Description

Attempts to convert a NetCDF-CF DSG Simple Geometry file into a sf data.frame.

Usage

```
read_geometry(nc_file)
```

Arguments

nc_file	character file path to the nc file to be read.

Value

sf data.frame containing spatial geometry of type found in the NetCDF-CF DSG file.

References

http://cfconventions.org/index.html

1. http://cfconventions.org/cf-conventions/cf-conventions.html#_features_and_feature_types

Examples

```
huc_eta_nc <- tempfile()
file.copy(system.file('extdata','example_huc_eta.nc', package = 'ncdfgeom'),
         huc_eta_nc, overwrite = TRUE)

vars <- ncmeta::nc_vars(huc_eta_nc)

hucPolygons <- sf::read_sf(system.file('extdata','example_huc_eta.json', package = 'ncdfgeom'))
plot(sf::st_geometry(hucPolygons))
names(hucPolygons)

hucPolygons_nc <- ncdfgeom::write_geometry(nc_file=huc_eta_nc,
                                           geom_data = hucPolygons,
                                           instance_dim_name = "station",
                                           variables = vars$name)
huc_poly <- read_geometry(huc_eta_nc)
plot(sf::st_geometry(huc_poly))
names(huc_poly)
```

---

read_timeseries_dsg *Read NetCDF-CF timeSeries featuretype*

Description

This function reads a timeseries discrete sampling geometry NetCDF file and returns a list containing the file's contents.

Usage

```
read_timeseries_dsg(nc_file, read_data = TRUE)
```

Arguments

nc_file	character file path to the nc file to be read.
read_data	logical whether to read metadata only or not.

Details

The current implementation checks several NetCDF-CF specific conventions prior to attempting to read the file. The Conventions and featureType global attributes are checked but not strictly required.

Variables with standard_name and/or cf_role of station_id and/or timeseries_id are searched for to indicate which variable is the 'timeseries identifier'. The function stops if one is not found.

All variables are introspected for a coordinates attribute. This attribute is used to determine which variables are coordinate variables. If none are found an attempt to infer data variables by time and timeseries_id dimensions is made.

The coordinates variables are introspected and their standard_names used to determine which coordinate they are. Lat, lon, and time are required, height is not.

Variables with a coordinates attribute are assumed to be the 'data variables'.

Data variables are traversed and their metadata and data content put into lists within the main response list.

See the timeseries vignette for more information.

Value

list containing the contents of the NetCDF file.

References

https://www.unidata.ucar.edu/software/netcdf-java/v4.6/reference/FeatureDatasets/CFpointImplement.html

---

write_attribute_data *Write attribute data to NetCDF-CF*

Description

Creates a NetCDF file with an instance dimension, and any attributes from a data frame. Use to create the start of a NetCDF-DSG file. One character length dimension is created long enough to contain the longest provided character string. This function does not implement any CF convention attributes or standard names. Any columns of class date will be converted to character.

Usage

```
write_attribute_data(
  nc_file,
  att_data,
  instance_dim_name = "instance",
  units = rep("unknown", ncol(att_data)),
  overwrite = FALSE
)
```

Arguments

nc_file	character file path to the nc file to be created. If adding to a file, it must already have the named instance dimension.
att_data	data.frame with instances as columns and attributes as rows.
instance_dim_name	character name for the instance dimension. Defaults to "instance"
units	character vector with units for each column of att_data. Defaults to "unknown" for all.
overwrite	boolean overwrite existing file? Will append if FALSE.

Examples

```
sample_data <- sf::st_set_geometry(sf::read_sf(system.file("shape/nc.shp",
                                                           package = "sf")),
                                   NULL)
example_file <-write_attribute_data(tempfile(), sample_data,
                                    units = rep("unknown", ncol(sample_data)))

try({
  ncdump <- system(paste("ncdump -h", example_file), intern = TRUE)
  cat(ncdump ,sep = "\n")
}, silent = TRUE)
```

---

write_geometry *Write geometries and attributes to NetCDF-CF*

Description

Creates a file with point, line or polygon instance data ready for the extended NetCDF-CF timeSeries featuretype format.

Will also add attributes if provided data has them.

Usage

```
write_geometry(
  nc_file,
  geom_data,
  instance_dim_name = NULL,
  variables = list()
)
```

Arguments

nc_file	character file path to the nc file to be created.
geom_data	sf data.frame with POINT, LINESTRING, MULTILINESTRING, POLYGON, or MULTIPOLYGON geometries. Note that three dimensional geometries are not supported.
instance_dim_name	character Not required if adding geometry to a NetCDF-CF Discrete Sampling Geometries timeSeries file. For a new file, will use package default – "instance" – if not supplied.
variables	character If a an existing netCDF files is provided, this list of variables that should be related to the geometries.

References

1. http://cfconventions.org/cf-conventions/cf-conventions.html

Examples

```
hucPolygons <- sf::read_sf(system.file('extdata','example_huc_eta.json', package = 'ncdfgeom'))

hucPolygons_nc <- ncdfgeom::write_geometry(nc_file=tempfile(),
                                           geom_data = hucPolygons)
try({
  ncdump <- system(paste("ncdump -h", hucPolygons_nc), intern = TRUE)
  cat(ncdump ,sep = "\n")
}, silent = TRUE)
```

---

write_timeseries_dsg *Write time series to NetCDF-CF*

Description

This function creates a timeseries discrete sampling geometry NetCDF file. It uses the orthogonal array encoding to write one data.frame per function call. This encoding is best suited to data with the same number of timesteps per instance (e.g. geometry or station).

Usage

```
write_timeseries_dsg(
  nc_file,
  instance_names,
  lats,
  lons,
  times,
  data,
  alts = NA,
  data_unit = "",
  data_prec = "double",
  data_metadata = list(name = "data", long_name = "unnamed data"),
  time_units = "days since 1970-01-01 00:00:00",
  instance_dim_name = "instance",
  dsg_timeseries_id = "instance_name",
 coordvar_long_names = list(instance = "Station Names", time = "time of measurement",
   lat = "latitude of the measurement", lon = "longitude of the measurement", alt =
    "altitude of the measurement"),
  attributes = list(),
  add_to_existing = FALSE,
  overwrite = FALSE
)
```

Arguments

nc_file	character file path to the nc file to be created.
instance_names	character or numeric vector of names for each instance (e.g. station or geometry) to be added to the file.
lats	numeric vector of latitudes
lons	numeric vector of longitudes
times	POSIXct vector of times. Must be of type POSIXct or an attempt to convert it will be made using as.POSIXct(times).
data	data.frame with each column corresponding to an instance. Rows correspond to time steps. nrow must be the same length as times. Column names must match instance names.
alts	numeric vector of altitudes (m above sea level) (Optional)
data_unit	character vector of data units. Length must be the same as number of columns in data parameter.
data_prec	character precision of observation data in NetCDF file. Valid options: 'short' 'integer' 'float' 'double' 'char'.
data_metadata	list A named list of strings: list(name='ShortVarName', long_name='A Long Name')
time_units	character units string in udunits format to use for time. Defaults to 'days since 1970-01-01 00:00:00'
instance_dim_name	the character name to use for the instance used in 'instance_names'

dsg_timeseries_id
: the character name to use for the instance used in the timeseries id

coordvar_long_names
: list values for long names on coordinate variables. Names should be 'instance', time', 'lat', 'lon', and 'alt.'

attributes
: list An optional list of attributes that will be added at the global level. See details for useful attributes.

add_to_existing
: boolean If TRUE and the file already exists, variables will be added to the existing file. See details for more.

overwrite
: boolean unless set to true, error if file exists.

Details

Suggested Global Variables: c(title = "title", abstract = "history", provider site = "institution", provider name ="source", description = "description")

Note regarding add_to_existing: add_to_existing = TRUE should only be used to add variables to an existing NetCDF discrete sampling geometry file. All other inputs should be the same as are already in the file. If the functions is called with add_to_existing=FALSE (the default), it will overwrite an existing file with the same name. The expected usage is to call this function repeatedly only changing the data, data_unit, data_prec and data_metadata inputs.

See the timeseries vignette for more information.

References

1. https://www.unidata.ucar.edu/software/netcdf-java/v4.6/reference/FeatureDatasets/CFpointImplement.html
2. http://cfconventions.org/cf-conventions/cf-conventions.html#_orthogonal_multidimensional_array_representation
3. http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/build/cf-conventions.html#time-series-data

Index