Package 'medfateland'

July 24, 2024

Type Package

Title Mediterranean Landscape Simulation

Version 2.4.5

Date 2024-07-22

Description Simulate forest hydrology, forest function and dynamics over landscapes [De Caceres et al. (2015) <doi:10.1016/j.agrformet.2015.06.012>]. Parallelization is allowed in several simulation functions and simulations may be conducted including spatial processes such as lateral water transfer and seed dispersal.

License GPL (>= 2)

URL https://emf-creaf.github.io/medfateland/

LazyLoad yes

Depends R (>= 3.4.0), medfate (>= 4.4.0)

Imports cli, ggplot2, dplyr, httr, jsonlite, lifecycle, methods, meteoland (>= 2.0.2), rlang, Rcpp (>= 0.12.12), parallel, sf, shiny, stars, terra, tidyterra, tidyr, tibble, stats

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

LinkingTo Rcpp, meteoland, medfate

Encoding UTF-8

NeedsCompilation yes

LazyData true

Config/testthat/edition 3

RoxygenNote 7.3.2

BugReports https://github.com/emf-creaf/medfateland/issues

Author Miquel De Cáceres [aut, cre],
Aitor Améztegui [aut] (<https://orcid.org/0000-0003-2006-1559>),
María González [aut] (<https://orcid.org/0000-0002-2227-8404>),
Núria Aquilué [aut],
Daniel Caviedes-Voullième [aut],
Mario Morales-Hernández [aut],
Mario Beltrán [ctb],

Rodrigo Balaguer-Romano [ctb] (<https://orcid.org/0000-0003-2808-6777>),
Roberto Molowny-Horas [ctb] (<https://orcid.org/0000-0003-2626-6379>)

Maintainer Miquel De Cáceres <miquelcaceres@gmail.com>

Repository CRAN

Date/Publication 2024-07-24 06:50:02 UTC

Contents

`add_forests` *Add forests*

Description

Creates and adds forest data to an `sf` object by reading from tree and shrub data tables

Usage

```
add_forests(
  x,
  tree_table = NULL,
  tree_mapping = NULL,
  shrub_table = NULL,
  shrub_mapping = NULL,
  merge_trees = TRUE,
  merge_shrubs = TRUE,
  SpParams = NULL,
  progress = FALSE
)
```

Arguments

x	An object of class sf with a valid CRS definition, and a column called 'id'.
tree_table	A data frame with tree records in rows and attributes in columns. Tree records can correspond to individual trees or groups of trees with an associated density.
tree_mapping	A named character vector to specify mappings of columns in tree_table into attributes of treeData. Accepted names (and the corresponding specifications for the columns in tree_table) are: • "id": Forest stand id. • "Species": Species code (should follow codes in SpParams). • "Species.name": Species name. In this case, the species code will be drawn by matching names with species names in SpParams. • "N": Tree density (in ind./ha). • "DBH": Diameter at breast height (in cm). • "Height": Tree height (in cm). • "plot.size": Plot size (in m2) to which each record refers to. This is used to calculate tree density (stems per hectare) when not supplied. • "Z50": Depth (in mm) corresponding to 50 percent of fine roots. • "Z95": Depth (in mm) corresponding to 95 percent of fine roots.
shrub_table	A data frame with shrub records in rows and attributes in columns. Records can correspond to individual shrubs (with crown dimensions and height) or groups of shrubs with an associated cover estimate.
shrub_mapping	A named character vector to specify mappings of columns in shrub_table into attributes of shrubData. Accepted names (and the corresponding specifications for the columns in shrub_table) are: • "id": Forest stand id. • "Species": Species code (should follow codes in SpParams). • "Species.name": Species name. In this case, the species code will be drawn by matching names with species names in SpParams. • "Cover": Shrub cover (in percent). • "D1": Shrub largest crown diameter (in cm).

- "D2": Shrub crown diameter orthogonal to the largest one (in cm).
- "Height": Shrub height (in cm).
- "plot.size": Plot size (in m2) to which each record refers to. This is used to calculate shrub cover when shrub data is given at the individual level.
- "Z50": Depth (in mm) corresponding to 50 percent of fine roots.
- "Z95": Depth (in mm) corresponding to 95 percent of fine roots.

`merge_trees` A logical flag to simplify tree cohorts by merging tree records in DBH classes (see `forest_mergeTrees`).

`merge_shrubs` A logical flag to simplify shrub cohorts by merging shrub records in height classes (see `forest_mergeShrubs`).

`SpParams` A data frame with species parameters (see `SpParamsMED`) from which valid species names are drawn.

`progress` A logical flag to include a progress bar while processing the data.

Details

The current implementation will replace existing forests of the indicated 'id' values.

Value

A modified object of class `sf` with column 'forest'.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

`impute_forests()`, `forest_mapWoodyTables`, `forest_mergeTrees`

Examples

```
# Load tree data
data(poblet_trees)

# Load species parameters
data(SpParamsMED)

# Define sf with three stands
cc <- rbind(c(1.0215, 41.3432),
            c(1.0219, 41.3443),
            c(1.0219, 41.3443))
d <- data.frame(lon = cc[,1], lat = cc[,2],
                id = c("POBL_CTL", "POBL_THI_BEF", "POBL_THI_AFT"))
x <- sf::st_as_sf(d, coords = c("lon", "lat"), crs = 4326)
x

# Define tree mapping
mapping <- c("id" = "Plot.Code", "Species.name" = "Species", "DBH" = "Diameter.cm")
```

```
# Read tree data (warnings are raised)
y_1 <- add_forests(x, tree_table = poblet_trees, tree_mapping = mapping, SpParams = SpParamsMED)

# Correct scientific name for downy oak and repeat to avoid losing tree records
poblet_trees$Species[poblet_trees$Species=="Quercus humilis"] <- "Quercus pubescens"
y_1 <- add_forests(x, tree_table = poblet_trees, tree_mapping = mapping, SpParams = SpParamsMED)

# Display summary of first forest
summary(y_1$forest[[1]], SpParamsMED)

# Add sampled plot surface and repeat reading to correct tree density
poblet_trees$PlotSurface <- 706.86
mapping <- c(mapping, "plot.size" = "PlotSurface")

y_2 <- add_forests(x, tree_table = poblet_trees, tree_mapping = mapping, SpParams = SpParamsMED)
summary(y_2$forest[[1]], SpParamsMED)

# Check forests (height is missing!)
check_forests(y_2)

# Estimate tree height using general allometric
poblet_trees$Height.cm <- 100 * 1.806*poblet_trees$Diameter.cm^0.518

#Modify mapping to include height and repeat
mapping <- c(mapping, "Height" = "Height.cm")

y_3 <- add_forests(x, tree_table = poblet_trees, tree_mapping = mapping, SpParams = SpParamsMED)
summary(y_3$forest[[1]], SpParamsMED)

# Final check
check_forests(y_3)
```

---

add_topography *Add topography and land cover*

Description

Initializes topography and land cover type for a set of target locations

Usage

```
add_topography(x, dem, progress = TRUE)

add_land_cover(
  x,
  land_cover_map,
  wildland = NULL,
  agriculture = NULL,
```

```
  rock = NULL,
  artificial = NULL,
  water = NULL,
  progress = TRUE
)

check_topography(x, filter_missing = FALSE)

check_land_cover(x, filter_missing = FALSE)
```

Arguments

x	An object of class sf
dem	A digital elevation model (class SpatRaster) with meters as units
progress	A logical flag to print console output
land_cover_map	An object of class SpatRaster of land cover type. If missing, all locations are considered 'wildland'.
wildland, agriculture, rock, artificial, water	Strings indicating the mapping from the legend of land_cover_map.
filter_missing	Boolean flag to filter locations with missing data

Details

The user should manually define the mapping of land cover classes in land_cover_map to the land cover types used in medfateland.

Value

Function add_topography() returns a modified object of class sf with columns:

- id: Numeric location identifiers (if not existing).
- elevation: Elevation above sea level (in m).
- slope: Slope (in degrees).
- aspect: Aspect (in degrees).
- land_cover_type: Land cover type.

Function add_land_cover() returns a modified object of class sf with new column:

- id: Numeric location identifiers (if not existing).
- land_cover_type: Land cover type.

See Also

impute_forests(), add_soilgrids()

Examples

```
# See package vignettes 'Preparing inputs'
```

create_fire_regime *Create fire regime*

Description

Defines an object containing fire regime parameters for simulations of forest dynamics.

Usage

```
create_fire_regime(annual_burned_area, sd_burned_area = NULL, doy = NULL)
```

Arguments

annual_burned_area	A named vector of burned area in hectares for simulation years.
sd_burned_area	A named vector of standard deviation (in log scale) of burned area. If specified, annual target to burn will be determined using a log-normal distribution with mean values given by annual_burned_area.
doy	A named integer vector with the day of the year (i.e. between 1 and 366) when fires will be simulated for each simulation year in annual_burned_area. If NULL fires will be simulated on the driest day (i.e. when vapor pressure deficit is largest).

Details

Names of annual_burned_area should be simulation years. If provided, sd_burned_area should be a vector of the same size as annual_burned_area and have the same names.

Value

A list with the supplied parameters

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

fire_regime_instance, fordyn_scenario, fordyn_spatial

Examples

```
# Fire regime with pre-defined burned area values
reg1 <- create_fire_regime(annual_burned_area = c("2002" = 1000, "2003" = 5000))

# Fire regime with log-normal distribution for burned area
reg2 <- create_fire_regime(annual_burned_area = c("2002" = 1000, "2003" = 5000),
                           sd_burned_area = c("2002" = 0.9, "2003" = 0.8))
```

`create_management_scenario` *Create management scenario*

Description

Defines a management scenario for simulations of forest dynamics

Usage

```
create_management_scenario(
  units,
  annual_demand_by_species = NULL,
  extraction_rate_by_year = NULL,
  default_management_arguments = NULL
)
```

Arguments

`units`	Number of management units. Alternatively, a data frame with management options (in columns) for a set of units (in rows). Options not specified witl be taken from defaults.
`annual_demand_by_species`	A vector or matrix of annual wood demand (m3) by medfate species names (or groups of species names). If empty, the scenario is 'bottom-up' (not based on demand). If a vector is supplied, the same wood demand is applied for all simulated years. If a matrix is supplied, each row should correspond to a different year.
`extraction_rate_by_year`	A vector of extraction rates (%) per year of the simulation, starting at the second year. If specified, the annual demand by species will be applied for the first year of the simulation, but it will be rescaled for the remaining years according to the growth observed and the desired extraction rates.
`default_management_arguments`	A list of arguments to be passed to the managementFunction. These arguments will be taken as defaults copied for all management units and can later be modified. If NULL, the result of calling function `defaultManagementArguments` will be taken.

Details

Three kinds of management scenarios are allowed:

1. `'bottom-up'` represents a scenario where forest stands belong to different management units, each of them having possibly distinct management prescriptions. However, there is no demand and the amount of extracted wood emerges from the interplay between forest dynamics and management prescriptions.

2. 'input_demand' represents a scenario where a certain amount of wood extraction is targeted for some species and each year. This requires deciding which stands will actually undergo thinning operations to fulfill the demand (stands managed following prescriptions that indicate final regeneration cuts are managed irrespective of demand).
3. 'input_rate' represents a scenario similar to the previous one but where total amount of wood targeted depends on (i.e. is a proportion of) the growth observed in previous year.

The kind of management scenario depends on the arguments supplied by the user when calling create_management_scenario (see examples). In all cases, management units need to be defined. Each management unit represents a group of forest stands following the same management prescriptions. Although the create_management_scenario function allows specifying the management arguments of each unit, the simulation of management scenarios also requires specifying, for each forest stand, to which management unit it belongs (see fordyn_scenario).

Value

A list with the following structure:

- scenario_type: Either 'bottom-up' (no demand is specified), 'input_demand' (annual species demand is specified), or 'input_rate' when extraction rates are also supplied.
- annual_demand_by_species: A vector of annual wood demand (m3) by species (or species groups) (for scenario_type 'bottom-up' or 'input_demand').
- extraction_rate_by_year: A vector of extraction rate values per year.
- units: A data frame with as many rows as units and management arguments as columns.

Author(s)

Miquel De Cáceres Ainsa, CREAF

Aitor Améztegui, UdL

See Also

fordyn_scenario, defaultManagementFunction, defaultPrescriptionsBySpecies, create_fire_regime

Examples

```
# A scenario with three management units and annual demand for two species
scen_1 <- create_management_scenario(3,  c("Quercus ilex" = 1000, "Pinus nigra" = 2000))

# A scenario like the former, but with total annual demand changing as a function of
# prescribed extraction rates (second and third years)
scen_2 <- create_management_scenario(3,
           c("Quercus ilex" = 1000, "Pinus nigra" = 2000),
           c("2002" = 30, "2003" = 50))

# A scenario with as many management units as rows in 'defaultPrescriptionsBySpecies'
# and not based on demand
data("defaultPrescriptionsBySpecies")
scen_3 <- create_management_scenario(defaultPrescriptionsBySpecies)
```

```
# A scenario with three management units and annual demand for one species group
# and a third species
scen_4 <- create_management_scenario(3,  c("Quercus ilex/Quercus pubescens" = 1000,
                                           "Pinus nigra" = 2000))
```

`defaultPrescriptionsBySpecies` *Default prescriptions by species*

Description

Default management prescriptions by species, defined according current practices in Catalonia (NE Spain)

Details

A data frame with 27 species (or species groups) in rows and management parameters in columns (defined in `defaultManagementArguments`)

Source

Mario Beltrán & Míriam Piqué. Forest Science and Technology Centre of Catalonia (CTFC)

See Also

`create_management_scenario`, `defaultManagementArguments`, `fordyn_scenario`

`default_dispersal_control` *Default control parameters for dispersal*

Description

Defines default control parameters for dispersal process

Usage

```
default_dispersal_control()
```

Value

A list with the following items:

- `distance_step` [= 25]: Distance step in meters.
- `maximum_dispersal_distance` [= 3000]: Maximum dispersal distance in meters.
- `min_percent` [= 1]: A minimum percent of seed bank to retain entry in `seedBank` element of `forest`.
- `stochastic_resampling` [= FALSE]: A flag to indicate that stochastic resampling of stands is performed.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

spwb_land, fordyn_scenario dispersal

Examples

```
default_dispersal_control()
```

`default_volume_function` *Default volume function*

Description

Example function for estimating wood volume (in m3/ha) from a tree table or forest object.

Usage

```
default_volume_function(x, SpParams = NULL)
```

Arguments

x	A data frame with columns 'DBH', 'Height' and 'N' or a forest object
SpParams	A data frame with species parameters (not used in the default function but will be called)

Details

Users should define their own functions taking into account that:

- Input should be named 'x' and consist of a tree table with tree records as rows and columns 'DBH' (cm), 'Height' (cm), and 'N' (ind./ha).
- Output should be a numeric vector of length equal to the number of tree records in 'x'

Value

A function amenable for wood volume estimation.

default_watershed_control *Default control parameters for watershed processes*

Description

Defines default control parameters for watershed processes

Usage

```
default_watershed_control(watershed_model = "tetis")
```

Arguments

watershed_model
: Hydrological model for watershed processes. Only "tetis" or "serghei" are accepted.

Value

A list with the following items:

- watershed_model: A string with the watershed model.
- weather_aggregation_factor [= 1]: An integer specifying the spatial aggregation for interpolated weather.
- tetis_parameters: A list of TETIS parameters with the following elements:
  - R_localflow [= 1.0]: Correction factor for soil hydraulic saturated conductivity (local vertical flows).
  - R_interflow [= 50.0]: Correction factor for soil hydraulic saturated conductivity (subsurface flow between grid cells).
  - R_baseflow [= 5.0]: Correction factor for bedrock hydraulic conductivity (groundwaterflow between grid cells).
  - n_interflow [= 1.0]: Exponent for the determination of interflow.
  - n_baseflow [= 1.0]: Exponent for the determination of baseflow.
  - num_daily_substeps [= 4]: Number of daily sub-steps for interflow calculations.
  - rock_max_infiltration [= 10]: Maximum infiltration rate (mm·day-1) for rock cells.
  - deep_aquifer_loss [= 0]: Daily loss rate from watershed aquifer towards a deeper aquifer not connected to outlets (mm·day-1).
- serghei_parameters: A list of SERGHEI parameters with the following elements:
  - input_dir [= ""]: Path to SERGHEI input files.
  - output_dir [= ""]: Path to SERGHEI output files.
  - force_equal_layer_widths [= FALSE]: A boolean flag to force equal layer widths (taken from the first soil element) in all soils.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

spwb_land

Examples

```
default_watershed_control()
```

`dispersal` *Seed production, dispersal and seed bank dynamics*

Description

Simulates seed bank mortality, seed production and dispersal among stands

Usage

```
dispersal(
  sf,
  SpParams,
  local_control = medfate::defaultControl(),
  distance_step = 25,
  maximum_dispersal_distance = 3000,
  min_percent = 1,
  stochastic_resampling = FALSE,
  progress = TRUE
)
```

Arguments

`sf`	An object of class `sf` using a UTM projection (to measure distances in m) and with the following columns: • `geometry`: Spatial geometry. • `forest`: Objects of class `forest`.
`SpParams`	A data frame with species parameters (see `SpParamsMED`).
`local_control`	A list of control parameters (see `defaultControl`)
`distance_step`	Distance step in meters.
`maximum_dispersal_distance`	Maximum dispersal distance in meters.
`min_percent`	A minimum percent of seed bank to retain entry in `seedBank` element of `forest`.
`stochastic_resampling`	A flag to indicate that stochastic resampling of stands is performed.
`progress`	Boolean flag to display progress information.

Details

The input 'sf' object has to be in a Universal Transverse Mercator (UTM) coordinate system (or any other projection using meters as length unit) for appropriate function behavior.

Dispersal kernel follows Clark et al. (1999) and potential seed donors (neighbors) are defined up to a given grid distance order. A maximum value of 100% seed bank refill is attained for species with seed production in all seed donors and the local cell.

Value

A list with forest objects (for wildland cover type) containing a modified seed bank

Author(s)

Miquel De Cáceres Ainsa, CREAF.

Roberto Molowny-Horas, CREAF.

References

Clark et al. (1999). Seed dispersal near and far: Patterns across temperate and tropical forests. Ecology 80(5):1475-1494

See Also

fordyn_land

Examples

```
data(example_watershed)
data(SpParamsMED)

# Transform to UTM31
example_watershed_utm31 <- sf::st_transform(example_watershed, crs = 32631)

# Estimate seed production and dispersal over the watershed
seedbank_list <- dispersal(example_watershed_utm31, SpParamsMED)

seedbank_list[[1]]

# Transform to UTM31
example_ifn_utm31 <- sf::st_transform(example_ifn, crs = 32631)

# Estimate seed production and dispersal over the set of forest inventory plots
seedbank_list <- dispersal(example_ifn_utm31, SpParamsMED)

seedbank_list[[1]]
```

example_ifn *Example of distributed forest inventory stands*

Description

An example of an coordinates, topography, forest and soil data corresponding to 30 forest inventory plots.

Format

The data format is that of an object `sf`

Source

- Soil data from SoilGrids global database (Hengl et al. 2017).
- Soil depth and depth to bedrock from Shangguan et al. (2017).
- Forest structure and composition from the Third Spanish Forest Inventory (IFN3).

See Also

`spwb_spatial`

example_watershed *Example of watershed*

Description

An example of an object of `sf` with data for a small catchment of 66 ha (0.66 km2) in Catalonia. Object `example_watershed_burnin` is the result of three years of burn-in period.

Format

The data format is that of an object `sf`

Source

- Watershed limits and channel network from the spanish Ministerio de Transición Ecológica y el Reto Demográfico.
- Elevation data at 30 m resolution from catalan Institut Cartogràfic i Geològic de Catalunya.
- Soil data from SoilGrids global database (Hengl et al. 2017).
- Soil depth and depth to bedrock from Shangguan et al. (2017).
- Bedrock hydraulic properties from Huscroft et al. (2018).
- Land cover data from Mapa Forestal de España 1:25000.
- Forest structure and composition from Mapa Forestal de España 1:25000 and the Third Spanish Forest Inventory (IFN3).

References

Hengl, T., Mendes De Jesus, J., Heuvelink, G.B.M., Gonzalez, M.R., Kilibarda, M., Blagotí, A., Shangguan, W., Wright, M.N., Geng, X., Bauer-Marschallinger, B., Guevara, M.A., Vargas, R., Macmillan, R.A., Batjes, N.H., Leenaars, J.G.B., Ribeiro, E., Wheeler, I., Mantel, S., Kempen, B., 2017. SoilGrids250m: Global Gridded Soil Information Based on Machine Learning. PLoS One 12, e0169748. doi:10.1371/journal.pone.0169748

Huscroft, J., Gleeson, T., Hartmann, J., Börker, J., 2018. Compiling and Mapping Global Permeability of the Unconsolidated and Consolidated Earth: GLobal HYdrogeology MaPS 2.0 (GLHYMPS 2.0). Geophys. Res. Lett. 45, 1897–1904. doi:10.1002/2017GL075860

Shangguan, W., Hengl, T., Mendes de Jesus, J., Yuan, H., Dai, Y., 2017. Mapping the global depth to bedrock for land surface modeling. J. Adv. Model. Earth Syst. 9, 65–88. doi:10.1002/2016MS000686

See Also

spwb_land

`extract_variables` *Landscape variables*

Description

Extract or estimate variables from landscape objects (class 'sf').

Usage

```
extract_variables(x, vars = "land_cover_type", SpParams = NULL, ...)

plot_variable(x, variable = "land_cover_type", SpParams = NULL, r = NULL, ...)
```

Arguments

x	An object of class sf with the appropriate columns.
vars	A string vector with the name of the variables to extract (see details).
SpParams	A data frame with species parameters (see SpParamsMED), required for most forest stand variables.
...	Additional arguments (not used).
variable	A string with the name of the variables to draw (see details).
r	An object of class SpatRaster, defining the raster topology.

Details

The following string values are available for `vars`.

Topography:

- `"elevation"`: Elevation in m.
- `"slope"`: Slope in degrees.
- `"aspect"`: Slope in degrees.
- `"land_cover_type"`: Land cover type.

Soil:

- `"soil_vol_extract"`: Total water extractable volume (mm).
- `"soil_vol_sat"`: Total water volume at saturation (mm).
- `"soil_vol_fc"`: Total water volume at field capacity (mm).
- `"soil_vol_wp"`: Total water volume at wilting point (mm).
- `"soil_vol_curr"`: Current total water volume (mm).
- `"soil_rwc_curr"`: Current soil relative water content (%).
- `"soil_rew_curr"`: Current soil relative extractable water (%).
- `"soil_theta_curr"`: Current soil moisture content (% vol.)
- `"soil_psi_curr"`: Current soil water potential (MPa).

Watershed:

- `"depth_to_bedrock"`: Depth to bedrock (m).
- `"bedrock_porosity"`: Bedrock porosity.
- `"bedrock_conductivity"`: Bedrock conductivity (m/day).
- `"aquifer_elevation"`: Aquifer elevation over bedrock (m).
- `"depth_to_aquifer"`: Depth to aquifer (m).
- `"aquifer"`: Aquifer volume (mm).
- `"snowpack"`: Snowpack water equivalent (mm).

Forest stand:

- `"basal_area"`: Basal area (m2/ha).
- `"tree_density"`: Tree density (ind/ha).
- `"mean_tree_height"`: Mean tree height (cm).
- `"dominant_tree_height"`: Dominant tree height (cm).
- `"dominant_tree_diameter"`: Dominant tree diameter (cm).
- `"quadratic_mean_tree_diameter"`: Quadratic mean tree diameter (cm).
- `"hart_becking_index"`: Hart-Becking index.
- `"leaf_area_index"`: Leaf area index (m2/m2).
- `"foliar_biomass"`: Foliar biomass (kg/m2).
- `"fuel_loading"`: Fine live fuel loading (kg/m2).
- `"shrub_volume"`: Shrub volume (m3/m2).

Value

Function `extract_variables()` returns an object of class `sf` with the desired variables. Function `plot_variables()` returns a ggplot object.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

See Also

`forest`, `soil`, `summary.forest`, `shinyplot_land`

Examples

```
# Load data and species parameters from medfate
data(example_ifn)
data(SpParamsMED)

# Calculate basal area and leaf area index
# for all forest stands
extract_variables(example_ifn, vars = c("basal_area", "leaf_area_index"),
                  SpParams = SpParamsMED)
```

---

fire_regime_instance *Fire regime instance*

Description

Applies a fire regime object over a set of landscape units to determine a fire realization

Usage

```
fire_regime_instance(sf, fire_regime)
```

Arguments

`sf`	An object of class `sf` with the following columns: • `geometry`: Spatial geometry. • `id`: Stand identifiers. • `represented_area_ha`: Area represented by each stand (in hectares). • `ignition_weights`: Relative weights to determine stands to be burned (optional).
`fire_regime`	A list of parameters defining the fire regime (see `create_fire_regime`).

Details

The function randomly determines the landscape units that will burn every year, depending on the specifications of the fire regime object. Users can define their own fire regime instances from other models (e.g. a fire landscape model) and then use those directly in functions fordyn_spatial or fordyn_scenario.

Value

An integer matrix specifying the day of the year of burning of each landscape unit for every year in the fire regime definition. Values are interpreted as follows:

- NA - No wildfire this year
- 0 - Wildfire will occur the driest day (i.e. the one with largest vapor pressure deficit).
- 1...366 - Day of the year when wildfire will occur

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

create_fire_regime, fordyn_spatial, fordyn_scenario

Examples

```
# Load example data
data("example_ifn")

# Assume that each stand represents 1km2 = 100 ha
example_ifn$represented_area_ha <- 100

# Define fire regime characteristics
reg1 <- create_fire_regime(c("2002" = 200, "2003" = 500))

# Create a fire regime instance
m1 <- fire_regime_instance(example_ifn, reg1)

# Check number of plots burned
colSums(!is.na(m1))

# Define fire regime characteristics with stochastic area burned
reg2 <- create_fire_regime(annual_burned_area = c("2002" = 200, "2003" = 500),
                           sd_burned_area = c("2002" = 0.4, "2003" = 0.5))

# Create a fire regime instance
m2 <- fire_regime_instance(example_ifn, reg2)

# Check number of plots burned
colSums(!is.na(m2))
```

fordyn_scenario *Scenario of forest dynamics*

Description

Evaluates forest dynamics over a landscape including climate and management scenarios

Usage

```
fordyn_scenario(
  sf,
  SpParams,
  meteo = NULL,
  management_scenario,
  volume_function = NULL,
  volume_arguments = NULL,
  local_control = defaultControl(),
  dispersal_control = default_dispersal_control(),
  dates = NULL,
  CO2ByYear = numeric(0),
  fire_regime = NULL,
  summary_function = NULL,
  summary_arguments = NULL,
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE,
  verbose = FALSE
)
```

Arguments

`sf` An object of class `sf` with the following columns:

- `geometry`: Spatial geometry.
- `id`: Stand identifiers.
- `elevation`: Elevation above sea level (in m).
- `slope`: Slope (in degrees).
- `aspect`: Aspect (in degrees).
- `forest`: Objects of class `forest`.
- `soil`: Objects of class `soil`.
- `state`: Objects of class `spwbInput` or `growthInput` (optional).
- `meteo`: Data frames with weather data (required if parameter `meteo = NULL`).
- `management_unit`: Management unit corresponding to each stand.
- `represented_area_ha`: Area represented by each stand in hectares.

- `ignition_weights`: Relative weights to determine stands to be burned. Optional, relevant when `fire_regime` is supplied only).
- `local_control`: A list of control parameters (optional). Used to override function parameter `local_control` for specific stands (values can be `NULL` for the remaining ones).

Alternatively, the user may supply the result of a previous call to `fordyn_scenario`, where to continue simulations.

`SpParams` A data frame with species parameters (see `SpParamsMED`).

`meteo` Meteorology data (see `fordyn_spatial`).

`management_scenario`
A list defining the management scenario (see `create_management_scenario`)

`volume_function`
A function accepting a forest object or a tree data table, and a species parameter table, as input and returning the wood volume (m3/ha) corresponding to each tree cohort. The function may accept additional arguments. If NULL, the default volume function is used (not recommended!).

`volume_arguments`
List with additional arguments for the volume function.

`local_control` A list of local model control parameters (see `defaultControl`).

`dispersal_control`
A list of dispersal control parameters (see `default_dispersal_control`). If NULL, then dispersal is not simulated.

`dates` A `Date` object with the days of the period to be simulated. If `NULL`, then the whole period of `meteo` is used.

`CO2ByYear` A named numeric vector with years as names and atmospheric CO2 concentration (in ppm) as values. Used to specify annual changes in CO2 concentration along the simulation (as an alternative to specifying daily values in `meteo`).

`fire_regime` A list of parameters defining the fire regime (see `create_fire_regime`) or a matrix representing a fire regime instance (see `fire_regime_instance`). If NULL, wildfires are not simulated. Details are given in `fordyn_spatial`.

`summary_function`
An appropriate function to calculate summaries from an object of class 'fordyn' (e.g., `summary.fordyn`).

`summary_arguments`
List with additional arguments for the summary function.

`parallelize` Boolean flag to try parallelization (will use all clusters minus one).

`num_cores` Integer with the number of cores to be used for parallel computation.

`chunk_size` Integer indicating the size of chunks to be sent to different processes (by default, the number of spatial elements divided by the number of cores).

`progress` Boolean flag to display progress information for simulations.

`verbose` Boolean flag to display additional console output.

Details

This function allows coordinating the dynamics of simulated forest stands via a management scenario defined at the landscape/regional level (see different kinds of scenarios and how to specify them in `create_management_scenario`).

The input 'sf' object has to be in a Universal Transverse Mercator (UTM) coordinate system (or any other projection using meters as length unit) for appropriate behavior of dispersal sub-model.

For each year to be simulated, the function determines which forest stands will be managed, possibly depending on the demand, and then calls function `fordyn_spatial` for one year (normally including parallelization). If the simulation of some stands results in an error, the function will try to restore the previous state of the forest stand for the next year steps. Finally, the function evaluates how much of the specified demand has been fulfilled and stores the results, including demand offsets to be applied the year after.

Management is implemented using the `defaultManagementFunction` in medfate, meaning that management parameters need to follow the structure of `defaultManagementArguments`

Details about the inclusion of fire regimes in simulations are explained in `fordyn_spatial`.

Value

An list of class 'fordyn_scenario' with the following elements:

- `result_sf`: An object of class 'sf' using a UTM projection and containing four elements:
  - `geometry`: Spatial geometry.
  - `id`: Stand id, taken from the input.
  - `tree_table`: A list of data frames for each simulated stand, containing the living trees at each time step.
  - `shrub_table`: A list of data frames for each simulated stand, containing the living shrub at each time step.
  - `dead_tree_table`: A list of data frames for each simulated stand, containing the dead trees at each time step.
  - `dead_shrub_table`: A list of data frames for each simulated stand, containing the dead shrub at each time step.
  - `cut_tree_table`: A list of data frames for each simulated stand, containing the cut trees at each time step.
  - `cut_shrub_table`: A list of data frames for each simulated stand, containing the cut shrub at each time step.
  - `summary`: A list of model output summaries for each simulated stand (if `summary_function` was not `NULL`).
- `result_volumes`: A data frame with initial, growth, extracted and final volumes (m3) by year. In demand-based scenarios volumes corresponding to species with demand are also included.
- `result_volumes_spp`: A data frame with growth and extracted volumes (m3) by species and year.
- `result_volumes_demand`: In demand-based scenarios target volumes are also included, a data frame with growth, target and extracted volumes (m3) by demand entity and year. .
- `next_sf`: An object of class 'sf' to continue simulations in subsequent calls to `fordyn_scenario`.
- `next_demand`: In demand-based scenarios, a list with information (i.e. demand offset by species and last volume growth) to modify demand in subsequent calls to `fordyn_scenario`.

Author(s)

Miquel De Cáceres Ainsa, CREAF

Aitor Améztegui, UdL

See Also

fordyn_spatial, create_management_scenario, dispersal

Examples

```
# Load example landscape data
data("example_ifn")

# Load example meteo data frame from package meteoland
data("examplemeteo")

# Load default medfate parameters
data("SpParamsMED")

# Creates scenario with one management unit and annual demand for P. nigra
scen <- create_management_scenario(1, c("Pinus nigra/Pinus sylvestris" = 2300))

# Assign management unit to all stands
example_ifn$management_unit <- 1

# Assume that each stand represents 1km2 = 100 ha
example_ifn$represented_area_ha <- 100

# Transform to UTM31 (necessary for dispersal)
example_ifn_utm31 <- sf::st_transform(example_ifn, crs = 32631)

# Subset three plots to speed-up calculations
example_subset <- example_ifn_utm31[31:33, ]

# Launch simulation scenario
fs_12 <- fordyn_scenario(example_subset, SpParamsMED, meteo = examplemeteo,
                         volume_function = NULL, management_scenario = scen,
                         parallelize = FALSE)
```

---

forest_parametrization

Landscape forest parametrization

Description

Utility functions to define forest inputs in a landscape:

- `impute_forests()` performs imputation of forest objects from a forest inventory using a forest map to match forest types and topography as covariates.
- `modify_forest_structure()` uses forest structure rasters supplied by the user to correct forest structure metrics.
- `check_forests()` checks that forests are defined and do not contain missing values in key tree/shrub attributes.

Usage

```
impute_forests(
  x,
  sf_fi,
  dem,
  forest_map,
  var_class = NA,
  max_distance_km = 100,
  replace_existing = FALSE,
  missing_class_imputation = FALSE,
  missing_class_forest = NULL,
  merge_trees = TRUE,
  merge_shrubs = TRUE,
  progress = TRUE
)

modify_forest_structure(
  x,
  structure_map,
  variable,
  map_var = NA,
  ratio_limits = NULL,
  minDBH = 7.5,
  progress = TRUE
)

check_forests(x, progress = FALSE)
```

Arguments

x	An object of class sf. If it contains a column named 'land_cover_type', imputation will be performed for locations whose land cover is "wildland". Otherwise, forest imputation is done for all locations. For structural corrections or when checking, x should already contain a column named 'forest' containing forest objects.
sf_fi	An object of class sf with forest inventory data column 'forest'.
dem	A digital elevation model (class SpatRaster) with meters as units
forest_map	An object of class SpatRaster or SpatVector with the forest class map
var_class	Variable name or index containing forest classes in forest_map. If missing the first column is taken.

max_distance_km
: Maximum distance, in km, for forest inventory plot imputation.

replace_existing
: A logical flag to force the replacement of existing forest objects, when present.

missing_class_imputation
: A logical flag to force imputation in locations where forest class is not defined. If missing_class_imputation = TRUE, imputation in those locations will be based on geographic and topographic criteria only.

missing_class_forest
: A forest object to be used for locations with missing class.

merge_trees
: A logical flag to simplify tree cohorts by merging tree records in DBH classes (see forest_mergeTrees).

merge_shrubs
: A logical flag to simplify shrub cohorts by merging shrub records in height classes (see forest_mergeShrubs).

progress
: A logical flag to print console output.

structure_map
: An object of class SpatRaster or SpatVector with a forest structural variable map

variable
: Structural variable to correct. See options in details.

map_var
: Variable name or index containing structural variable in 'structure_map'. If missing the first column is taken.

ratio_limits
: Limits for ratio of variable in corrections, used to avoid outliers.

minDBH
: Minimum diameter for stand metric calculation. If minDBH > 0 then those stands with smaller trees will not be corrected because of the missing stand metric. A special case occurs for correction following basal area (see details).

Details

Function impute_forests() performs imputation of forest inventory plots on target locations provided that they correspond to the same forest class, defined in the input forest map, and are geographically closer than a distance threshold (max_distance_km). Among the multiple stands that can have fulfill these two requirements, the function chooses the one that has the most similar elevation and position in the N-to-S slopes (i.e. the product of the cosine of aspect and slope). Both topographic features are standardized to zero mean and unit standard deviation (using the supplied digital elevation model to calculate those metrics), to make their weight on the imputation equal. This imputation method will be more or less successful depending on the resolution of forest classes and the number of forest inventory plots available for each of them. Additionally, tree and shrub cohorts can be simplified after imputation (merge_trees and merge_shrubs), to reduce the number of records (and hence, speed-up simulations).

Function modify_forest_structure() can be used to modify specific structure variables of the imputed forests building on rasters supplied by the user (typically from aerial or satellite LiDAR products). For any given metric, the function will calculate the ratio of the structure metric between the target forest object (see stand_basalArea) and the input map in the target location. Options for structural variables are the following:

- mean_tree_height: Should contain values in cm. Corrects tree heights and diameters (assuming a constant diameter-height relationship).

- `dominant_tree_height`: Should contain values in cm. Corrects tree heights and diameters (assuming a constant diameter-height relationship).
- `tree_density`: Should contain values in individuals per hectare. Corrects tree density.
- `basal_area`: Should contain values in squared meters per hectare (m2/ha). Corrects tree density. Forests that
- `mean_shrub_height`: Should contain values in cm. Corrects shrub cover.

Locations where the metric value in the map is missing are left unmodified. The same happens if metric value is zero, to avoid division by zero. A special case occurs for correction of basal area. In that case, if there are no trees larger than `minDBH` but structural map indicates positive values of basal area, DBH values will be set to minDBH, and correction of basal area will be performed.

Function `check_forest()` checks first that `forest` objects are defined in "wildland" locations. Then, it looks for missing data in tree or shrub attributes required for simulations. The function does not modify the data.

Value

Functions `impute_forests()` and `modify_forest_structure()` return a modified object of class `sf`. Function `check_forests()` returns an invisible data frame with columns indicating missing forest data and missing values in tree or shrub parameters.

Author(s)

Miquel De Cáceres Ainsa, CREAF

Rodrigo Balaguer-Romano, CREAF

See Also

`add_topography()`, `add_forests()`, `add_soilgrids()`, `forest_mergeTrees`

Examples

```
# See package vignettes 'Preparing inputs'
```

`initialize_landscape` *Initialization of model inputs for spatially-distributed forest stands*

Description

Initializes state for local models `spwb` or `growth`.

Usage

```
initialize_landscape(
  x,
  SpParams,
  local_control,
  model = "spwb",
  simplify = FALSE,
  replace = FALSE,
  progress = TRUE
)
```

Arguments

x	An object of class sf with the following columns: • geometry: Spatial geometry. • forest: Objects of class forest. • soil: Objects of class soil or data frames of physical properties. • land_cover_type: Land cover type of each grid cell (values should be 'wildland' or 'agriculture'). • crop_factor: Crop evapo-transpiration factor. Only required for 'agriculture' land cover type. • local_control: A list of control parameters (optional). Used to override function parameter local_control for specific cells (values can be NULL for the remaining ones).
SpParams	A data frame with species parameters (see SpParamsMED).
local_control	A list of control parameters (see defaultControl).
model	A string to indicate the model, either "spwb" or "growth".
simplify	Boolean flag to simplify forest to the tree and shrub cohorts with largest leaf area index. The leaf area index of the whole tree (respectively, shrub) layer will be attributed to the selected cohort. See function forest_reduceToDominant.
replace	Boolean flag to replace existing initialized states
progress	Boolean flag to display progress information.

Details

Initialization is dealt automatically when calling simulation functions spwb_spatial, growth_spatial, spwb_spatial_day or growth_spatial_day. However, function initialize_landscape allows separating initialization from model simulations.

Option simplify has been implemented to allow simplification of forests to tree/shrub dominant cohorts during watershed simulations where focus is on runoff (e.g. calibration of watershed parameters or burnin periods). Elements identified as result_cell will not be simplified.

Value

Replaces or adds column 'state' whose elements are spwbInput or growthInput objects and returns the modified object of class 'sf'.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

spwb_spatial, spwb_spatial_day, update_landscape

Examples

```
# Load example landscape data
data("example_ifn")

# Load example meteo data frame from package meteoland
data("examplemeteo")

# Load default medfate parameters
data("SpParamsMED")

# Define local control parameters using function in medfate
local_control <- defaultControl()

# If necessary, change defaults

# Initialize state for 'spwb' simulations
example_ifn_init <- initialize_landscape(example_ifn, SpParamsMED,
                                         local_control = local_control,
                                         model = "spwb")
```

---

landscape_summary *Forest and soil summaries over space*

Description

Functions to calculates a summary function for the forest or soil of all spatial elements in an object of class sf containing landscape information.

Usage

```
landscape_summary(
  object,
  name,
  summary_function,
  ...,
  unlist = FALSE,
  progress = FALSE
)
```

Arguments

object	An object of class sf.
name	A string of the element to summarize: "forest", "soil" or "state".
summary_function	A function that accepts objects of class forest, soil or model input objects, respectively.
...	Additional arguments to the summary function.
unlist	Logical flag to try converting the summaries into different columns
progress	Boolean flag to display progress information

Value

An object of class sf containing the calculated statistics. If unlist = FALSE column 'summary' is a list with summaries for each element. If unlist = TRUE different columns are returned instead, one per variable given in the summary function.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

See Also

forest, soil, summary.forest

Examples

```
# Load plot data and species parameters from medfate
data(example_ifn)

# Load default medfate parameters
data("SpParamsMED")

# Apply forest summary function
landscape_summary(example_ifn, "forest", summary.forest, SpParamsMED)
```

---

parse_forestable *Parse forestable*

Description

Transforms a data frame or sf object issued from package forestables into an sf object for simulations with medfateland.

Usage

```
parse_forestable(
  x,
  keepSpeciesCodes = TRUE,
  filterDeadTrees = TRUE,
  filterCutTrees = TRUE,
  minimumTreeDBH = 0.1,
  progress = FALSE
)
```

Arguments

x	A data frame or sf object issued from package forestables.
keepSpeciesCodes	Keeps forest inventory species codes.
filterDeadTrees	If TRUE, filters out dead trees (Spanish forest inventory IFN3 or IFN4).
filterCutTrees	If TRUE, filters out cut trees (Spanish forest inventory IFN3 or IFN4).
minimumTreeDBH	Minimum DBH for keeping a tree record.
progress	A logical flag to include a progress bar while processing the data.

Details

This function retrieves the following information from the forestables object:

- Id unique code, survey year, non-unique plot code and country.
- Plot location. Output geometry is always points in WGS 84. Note that exact coordinates are not normally given in forest inventory data.
- Elevation, slope and aspect, whenever available
- Tree and understory data. The function will create a column `forest` with this information.

Value

An sf object including a 'forest' column

---

plot_summary *Displays spatial simulation summaries*

Description

Produces graphical output of the summaries of a simulation models

Usage

```
plot_summary(x, variable, date, r = NULL, ...)
```

Arguments

x	An object of class sf, with simulation summaries.
variable	The variable to be drawn.
date	The date of the summary to be plotted.
r	An object of class SpatRaster, defining the raster topology.
...	Additional parameters (passed to scale definition, such as limits).

Details

Appropriate values for x can originate from calls to simulation_summary. Alternatively, if summary functions were specified at the time of performing simulations, the result of the spatial simulation function (e.g. spwb_spatial) will already contain the summaries. A special case is made for spwb_land and growth_land, that are accepted inputs as x, because its element 'sf' is used.

Value

An object of class ggplot.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

See Also

spwb_spatial, simulation_summary

---

shinyplot_land *Shiny app with interactive plots and maps*

Description

Creates a shiny app with interactive plots for spatial inputs and simulation results

Usage

```
shinyplot_land(x, SpParams = NULL, r = NULL)
```

Arguments

x	The object of class 'sf' containing information to be drawn (see details). Alternatively, an object of class 'spwb_land', 'growth_land' or 'fordyn_land'.
SpParams	A data frame with species parameters (see SpParamsMED), required for most forest stand variables.
r	An object of class SpatRaster, defining the raster topology.

Details

Only run this function in interactive mode. The shiny app can be used to display spatial inputs or simulation results.

Spatial inputs: This is the case if the user supplies an object of class `sf` with simulation inputs.

Simulation result summaries: This is the case if the user supplies an object of class `sf` with simulation summaries. Available plots depend on the summary function used to create the result summaries.

Value

An object that represents the shiny app

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

`plot_summary`, `extract_variables`

---

simulation_summary *Summarizes spatial simulation results*

Description

Creates spatial objects containing summaries of simulations

Usage

```
simulation_summary(object, summary_function, ...)
```

Arguments

`object`	An object of class 'sf' simulation results (e.g. the result of calling `spwb_spatial`).
`summary_function`	The summary function to be executed on simulation results (see details).
`...`	Additional parameters to the summary function.

Details

The function supplied should take as input an object of local simulation function, i.e. `spwb`, `growth`, or `fordyn`. The output should be a matrix with dates as rows and variables in columns. An example of suitable function is `summary.spwb`.

Value

An object of class sf, with the following two elements:

- geometry: Spatial geometry.
- id: Stand id, taken from the input.
- summary: A list of model output summaries for each simulated location.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

See Also

spwb_spatial, plot_summary

---

soil_parametrization *Landscape soil parametrization*

Description

Function add_soilgrids fills column 'soil' with physical soil characteristics drawn from SoilGrids 2.0 (Hengl et al. 2017; Poggio et al. 2021). Function modify_soils modifies soil definition according to soil depth and depth to bedrock information. Function check_soils verifies that soil data does not contain missing values for key variables and, if so, assigns default values.

Usage

```
add_soilgrids(
  x,
  soilgrids_path = NULL,
  widths = NULL,
  replace_existing = TRUE,
  progress = TRUE
)

modify_soils(
  x,
  soil_depth_map = NULL,
  depth_to_bedrock_map = NULL,
  regolith_rfc = 97.5,
  full_rock_filling = TRUE,
  progress = TRUE
)

check_soils(
  x,
  check_equal_layers = FALSE,
```

```
  fill_missing = FALSE,
  default_values = c(clay = 25, sand = 25, bd = 1.5, rfc = 25),
  progress = FALSE
)
```

Arguments

x	An object of class sf with a valid CRS definition. If it contains a column called 'land_cover_type', soils will be retrieved for "agriculture" and "wildland" cover types only. Otherwise, soils are retrieved for all locations. For functions modify_soils or check_soils, x should already contain a column named "soil".
soilgrids_path	Path to SoilGrids rasters (see details). If missing, the SoilGrids REST API (https://rest.isric.org) will be queried.
widths	A numeric vector indicating the desired layer widths, in *mm*. If NULL the default soil grids layer definition is returned.
replace_existing	A logical flag to force the replacement of existing soil data, when already present
progress	A logical flag to include a progress bar while processing the output of the query to the SoilGrids REST API.
soil_depth_map	An object of class SpatRaster or SpatVector with the soil depth (in *mm*) values.
depth_to_bedrock_map	An object of class SpatRaster or SpatVector with depth to bedrock (in *mm*) values.
regolith_rfc	Rock fragment content, in percent volume, between soil depth and 200cm depth (or lower depths, if modified via widths).
full_rock_filling	Logical flag to modify rock fragment content in all soil layers with according to distance to soil depth.
check_equal_layers	Logical flag to test whether soils have the same number of layers.
fill_missing	Logical flag to fill missing values in key parameters with defaults.
default_values	Vector of default values for locations with missing data.

Details

If soilgrids_path = NULL the function connects with the SoilGrids REST API (https://rest.isric.org) to retrieve the soil physical and chemical characteristics for a site (Hengl *et al*. 2007; Poggio et al. 2021), selected by its coordinates. Also, in case the depths are not the default ones in the SoilGrids API, the function uses averages the values of soil grid layers depending on the overlap between soil layer definitions. Unfortunately, SoilGrids REST API queries are limited to a few points.

If soilgrids_path != NULL the function will read SoilGrid rasters from the file disk. Folders need to be defined for each variable ("sand", "clay", "soc", "bdod", "cfvo" and "nitrogen"). File paths from soilgrids_path should be named:

var/*var_layer*_mean.tif

where *var* is one of the above and *layer* is "0-5cm", "5-15cm", "15-30cm", "30-60cm", "60-100cm" or "100-200cm"

SoilGrids does not provide soil depth estimates. Function `modify_soils` is designed to adjust soil depths according to available information. When `soil_depth_map` is provided, the function adjusts rock fragment content of layers below soil depth with the value of `regolith_rfc`. When `depth_to_bedrock_map` is provided, the function truncates the total depth of the soil definition to the depth to bedrock. If regional maps of soil depth are not available, users are recommended to resort on Shangguan et al (2017).

Value

A modified object of class `sf` with column 'soil'.

Author(s)

Víctor Granda, EMF-CREAF

Miquel De Cáceres Ainsa, EMF-CREAF

References

Hengl T, Mendes de Jesus J, Heuvelink GBM, Ruiperez Gonzalez M, Kilibarda M, Blagotić A, et al. (2017) SoilGrids250m: Global gridded soil information based on machine learning. PLoS ONE 12(2): e0169748. doi:10.1371/journal.pone.0169748.

Poggio L, de Sousa LM, Batjes NH, Heuvelink GBM, Kempen B, Ribeiro E, Rossiter D (2021). SoilGrids 2.0: producing soil information for the globe with quantified spatial uncertainty. SOIL 7, 217-240. doi: 10.5194/soil-7-217-2021

Shangguan W, Hengl T, Mendes de Jesus J, Yuan H, Dai J (2017). Mapping the global depth to bedrock for land surface modeling. Journal of Advances in Modeling Earth Systems 9: 65-88. doi: 10.1002/2016MS000686

See Also

`add_topography()`, `impute_forests()`, `soil`, `defaultSoilParams`

Examples

```
library(sf)
x <- st_sf(geometry = st_sfc(st_point(c(-5.6333, 42.6667)), crs = 4326))
x_soil <- add_soilgrids(x, widths = c(300, 700, 1000))
x_soil
# See more complete examples in package vignettes 'Preparing inputs'
```

spwb_land *Watershed simulations*

Description

Functions to perform simulations on a watershed described by a set of connected grid cells.

- Function spwb_land implements a distributed hydrological model that simulates daily local water balance, from spwb_day, on grid cells of a watershed while accounting for overland runoff, subsurface flow and groundwater flow between cells.
- Function growth_land is similar to spwb_land, but includes daily local carbon balance, growth and mortality processes in grid cells, provided by growth_day.
- Function fordyn_land extends the previous two functions with the simulation of management, seed dispersal, recruitment and resprouting.

Usage

```
spwb_land(
  r,
  sf,
  SpParams,
  meteo = NULL,
  dates = NULL,
  CO2ByYear = numeric(0),
  summary_frequency = "years",
  local_control = defaultControl(soilDomains = "single"),
  watershed_control = default_watershed_control(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)

growth_land(
  r,
  sf,
  SpParams,
  meteo = NULL,
  dates = NULL,
  CO2ByYear = numeric(0),
  summary_frequency = "years",
  local_control = medfate::defaultControl(soilDomains = "single"),
  watershed_control = default_watershed_control(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
```

```
  progress = TRUE
)

fordyn_land(
  r,
  sf,
  SpParams,
  meteo = NULL,
  dates = NULL,
  CO2ByYear = numeric(0),
  local_control = medfate::defaultControl(soilDomains = "single"),
  watershed_control = default_watershed_control(),
  dispersal_control = default_dispersal_control(),
  management_function = NULL,
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)

cell_neighbors(sf, r)

## S3 method for class 'spwb_land'
summary(object, ...)

## S3 method for class 'growth_land'
summary(object, ...)
```

Arguments

r	An object of class SpatRaster, defining the raster topology.
sf	An object of class sf with the following columns: • geometry: Spatial point geometry corresponding to cell centers. • elevation: Elevation above sea level (in m). • slope: Slope (in degrees). • aspect: Aspect (in degrees). • land_cover_type: Land cover type of each grid cell (values should be 'wildland', 'agriculture', 'rock', 'artificial' or 'water'). • forest: Objects of class forest. • soil: Objects of class soil or data frames of physical properties. • state: Objects of class spwbInput or growthInput (optional). • meteo: Data frames with weather data (required if parameter meteo = NULL). • crop_factor: Crop evapo-transpiration factor. Only required for 'agriculture' land cover type. • local_control: A list of control parameters (optional). Used to override function parameter local_control for specific cells (values can be NULL for the remaining ones).

- snowpack: An optional numeric vector with the snow water equivalent content of the snowpack in each cell (in mm). If missing it will be initialized to zero.
- management_arguments: Lists with management arguments (optional, relevant for fordyn_land only).
- result_cell: A logical indicating that local model results are desired (optional, relevant for spwb_land and growth_land only). Model results are only produced for wildland and agriculture cells.

When using TETIS watershed model, the following columns are also REQUIRED:

- depth_to_bedrock: Depth to bedrock (mm).
- bedrock_conductivity: Bedrock (saturated) conductivity (in m·day-1).
- bedrock_porosity: Bedrock porosity (the proportion of pore space in the rock).

When using TETIS watershed model, the following columns are OPTIONAL:

- aquifer: A numeric vector with the water content of the aquifer in each cell (in mm). If missing, it will be initialized to zero.
- deep_aquifer_loss: A numeric vector with the maximum daily loss to a deeper aquifer (in mm·day-1). If missing all cells take their value from deep_aquifer_loss in default_watershed_control

SpParams — A data frame with species parameters (see SpParamsMED).

meteo — Input meteorological data (see spwb_spatial and details).

dates — A Date object describing the days of the period to be modeled.

CO2ByYear — A named numeric vector with years as names and atmospheric CO2 concentration (in ppm) as values. Used to specify annual changes in CO2 concentration along the simulation (as an alternative to specifying daily values in meteo).

summary_frequency — Frequency in which cell summary will be produced (e.g. "years", "months", ...) (see cut.Date). In fordyn_land summaries are always produced at monthly resolution.

local_control — A list of control parameters (see defaultControl) for function spwb_day or growth_day. By default, parameter soilDomains is set to "single", meaning a single-domain Richards model.

watershed_control — A list of watershed control parameters (see default_watershed_control). Importantly, the sub-model used for lateral water flows - either Francés et al. (2007) or Caviedes-Voullième et al. (2023) - is specified there.

parallelize — Boolean flag to try parallelization (see details).

num_cores — Integer with the number of cores to be used for parallel computation (by default it will use all clusters minus one).

chunk_size — Integer indicating the size of chunks to be sent to different processes (by default, the number of spatial elements divided by the number of cores).

progress — Boolean flag to display progress information for simulations.

`dispersal_control`
: A list of dispersal control parameters (see `default_dispersal_control`). If NULL, then dispersal is not simulated.

`management_function`
: A function that implements forest management actions (see `fordyn`). of such lists, one per spatial unit.

`object`
: An object of class `spwb_land` or `groth_land`

`...`
: Additional parameters for summary functions

Details

The default `soilDomains = "single"` means that vertical bulk soil flows are simulated using a single permeability domain with Richards equation.

Two sub-models are available for lateral water transfer processes (overland flow, sub-surface flow, etc.), either "TETIS" (similar to Francés et al. 2007) or "SERGHEI" (Caviedes-Voullième et al. 2023).

IMPORTANT: medfateland needs to be compiled along with SERGHEI model in order to launch simulations with using this distributed hydrological model.

When running `fordyn_land`, the input 'sf' object has to be in a Universal Transverse Mercator (UTM) coordinate system (or any other projection using meters as length unit) for appropriate behavior of dispersal sub-model.

Parallel computation is only recommended for watersheds with large number of grid cells (e.g. > 10,000 when using `transpirationMode = "granier"`). In watershed with a small number of cells, parallel computation can result in larger processing times than sequential computation, due to the communication overload.

When dealing with large data sets, weather data included in the 'sf' object will likely be very data hungry. In those cases, it is recommended to resort on weather interpolation (see `spwb_spatial`). Weather interpolation can be done using a coarser resolution than that of raster 'r', by changing the watershed control parameter called 'weather_aggregation_factor' (see `default_watershed_control`).

Value

Functions `spwb_land`, `growth_land` and `fordyn_land` return a list of class of the same name as the function with the following elements:

- `watershed_control`: A list with input control parameters.
- `sf`: An object of class `sf`, similar to the output of `spwb_spatial`, with the following columns:
  - `geometry`: Spatial geometry.
  - `state`: A list of model input objects for each simulated stand.
  - `aquifer`: A numeric vector with the water volume in the aquifer of each cell.
  - `snowpack`: A numeric vector with the snowpack water equivalent volume of each cell.
  - `summary`: A list of cell summaries, containing the following variables:
    - `MinTemperature`: Minimum temperature (degrees Celsius).
    - `MaxTemperature`: Maximum temperature (degrees Celsius).
    - `PET`: Potential evapotranspiration (in mm).

    - Rain: Rainfall (in mm).
    - Snow: Snowfall (in mm).
    - Snowmelt: Snow melt (in mm).
    - Interception: Rainfall interception (in mm).
    - NetRain: Net rainfall, i.e. throughfall, (in mm).
    - Infiltration: The amount of water infiltrating into the soil (in mm).
    - InfiltrationExcess: The amount of water exceeding the soil infiltration capacity (in mm).
    - SaturationExcess: The amount of water that reaches the soil surface because of soil saturation (in mm).
    - Runon: The amount of water reaching the cell via surface runon (in mm).
    - Runoff: The amount of water exported from the cell via surface runoff (in mm).
    - DeepDrainage: The amount of water draining from soil to the aquifer via deep drainage (in mm).
    - CapillarityRise: Water entering the soil via capillarity rise (mm) from the water table.
    - SoilEvaporation: Bare soil evaporation (in mm).
    - Transpiration: Woody plant transpiration (in mm).
    - HerbTranspiration: Herbaceous transpiration (in mm).
    - InterflowInput: The amount of water that reaches the soil of the cell from adjacent cells via subsurface flow (in mm).
    - InterflowOutput: The amount of water that leaves the soil of the cell towards adjacent cells via subsurface flow (in mm).
    - InterflowBalance: The balance of water circulating via subsurface flow (in mm).
    - BaseflowInput: The amount of water that reaches the aquifer of the cell from adjacent cells via groundwater flow (in mm).
    - BaseflowOutput: The amount of water that leaves the aquifer of the cell towards adjacent cells via groundwater flow (in mm).
    - BaseflowBalance: The balance of water circulating via groundwater flow (in mm).
    - AquiferExfiltration: The amount of water of the cell that generates surface runoff due to the aquifer reaching the soil surface (in mm).
    - SWE: Snow water equivalent (in mm) of the snowpack.
    - RWC: Soil relative water content with respect to field capacity (in percent).
    - SoilVol: Soil water volume integrated across vertical layers (in mm).
    - WTD: Saturated soil water table depth (in mm from surface).
    - DTA: Depth to aquifer (in m from surface).
  - result: A list of cell detailed results (only for those indicated in the input), with contents depending on the local model.
  - outlet: A logical vector indicating outlet cells.

In function fordyn_land the sf object contains additional columns:

  - forest: A list of forest objects for each simulated stand, to be used in subsequent simulations (see update_landscape).
  - management_arguments: A list of management arguments for each simulated stand, to be used in subsequent simulations (see update_landscape).

  - `tree_table`: A list of data frames for each simulated stand, containing the living trees at each time step.
  - `shrub_table`: A list of data frames for each simulated stand, containing the living shrub at each time step.
  - `dead_tree_table`: A list of data frames for each simulated stand, containing the dead trees at each time step.
  - `dead_shrub_table`: A list of data frames for each simulated stand, containing the dead shrub at each time step.
  - `cut_tree_table`: A list of data frames for each simulated stand, containing the cut trees at each time step.
  - `cut_shrub_table`: A list of data frames for each simulated stand, containing the cut shrub at each time step.

- `watershed_balance`: A data frame with as many rows as days and where columns are components of the water balance at the watershed level (i.e., rain, snow, interception, infiltration, soil evaporation, plant transpiration, ...).
- `watershed_soil_balance`: A data frame with as many rows as days and where columns are components of the water balance at the watershed level restricted to those cells with a soil definition.
- `outlet_export_m3s`: A matrix with daily values of runoff (in m3/s) reaching each of the outlet cells of the landscape. Each outlet drains its own subset of cells, so the overall watershed export corresponds to the sum of row values.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

Maria González-Sanchís, Universitat Politecnica de Valencia.

Daniel Caviedes-Voullième, Forschungszentrum Julich.

Mario Morales-Hernández, Universidad de Zaragoza.

References

Francés, F., Vélez, J.I. & Vélez, J.J. (2007). Split-parameter structure for the automatic calibration of distributed hydrological models. Journal of Hydrology, 332, 226–240.

Caviedes-Voullième, D., Morales-Hernández, M., Norman, M.R. & Ogzen-Xian, I. (2023). SERGHEI (SERGHEI-SWE) v1.0: a performance-portable high-performance parallel-computing shallow-water solver for hydrology and environmental hydraulics. Geoscientific Model Development, 16, 977-1008.

See Also

`default_watershed_control`, `initialize_landscape`, `spwb_land_day`, `spwb_day`, `growth_day`, `spwb_spatial`, `fordyn_spatial`, `dispersal`

Examples

```
# Load example watershed data
data("example_watershed")

# Set crop factor
example_watershed$crop_factor <- NA
example_watershed$crop_factor[example_watershed$land_cover_type=="agriculture"] <- 0.75

# Set request for daily model results in cells number 3, 6 (outlet) and 9
example_watershed$result_cell <- FALSE
example_watershed$result_cell[c(3,6,9)] <- TRUE

# Get bounding box to determine limits
b <- sf::st_bbox(example_watershed)
b

# Define a raster topology, using terra package,
# with the same CRS as the watershed. In this example cells have 100 m side.
# Coordinates in the 'sf' object are assumed to be cell centers
r <-terra::rast(xmin = 401380, ymin = 4671820, xmax = 402880, ymax = 4672620,
                nrow = 8, ncol = 15, crs = "epsg:32631")

# Load example meteo data frame from package meteoland
data("examplemeteo")

# Load default medfate parameters
data("SpParamsMED")

# Set simulation period
dates <- seq(as.Date("2001-01-01"), as.Date("2001-03-31"), by="day")

# Watershed control parameters (TETIS model; Frances et al. 2007)
ws_control <- default_watershed_control("tetis")

# Launch simulations
res <- spwb_land(r, example_watershed, SpParamsMED, examplemeteo,
                 dates = dates, summary_frequency = "month",
                 watershed_control = ws_control)

# Print a summary of water balance components
summary(res)

# Option 'simplify = TRUE' in initialization, may be useful to speed up calculations
example_simplified <- initialize_landscape(example_watershed, SpParams = SpParamsMED,
                                   local_control = defaultControl(soilDomains = "single"),
                                            simplify = TRUE)

# Launch simulations over simplified landscape (should be considerably faster)
res_simplified <- spwb_land(r, example_simplified, SpParamsMED, examplemeteo,
                            dates = dates, summary_frequency = "month",
                            watershed_control = ws_control)
```

spwb_land_day *One-day watershed simulations*

Description

Functions to perform one-day simulations on a watershed described by a set of connected grid cells.

- Function spwb_land_day implements a distributed hydrological model that simulates daily local water balance, from spwb_day, on grid cells of a watershed while accounting for overland runoff, subsurface flow and groundwater flow between cells.
- Function growth_land_day is similar to spwb_land_day, but includes daily local carbon balance, growth and mortality processes in grid cells, provided by growth_day.

Usage

```
spwb_land_day(
  r,
  sf,
  SpParams,
  meteo = NULL,
  date = NULL,
  local_control = medfate::defaultControl(),
  watershed_control = default_watershed_control(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)

growth_land_day(
  r,
  sf,
  SpParams,
  meteo = NULL,
  date = NULL,
  local_control = medfate::defaultControl(),
  watershed_control = default_watershed_control(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)
```

Arguments

r	An object of class SpatRaster, defining the raster topology.
sf	An object of class sf as described in spwb_land.
SpParams	A data frame with species parameters (see SpParamsMED).
meteo	Input meteorological data (see spwb_spatial and details).
date	A string with the date to be simulated.
local_control	A list of control parameters (see defaultControl) for function spwb_day or growth_day.
watershed_control	A list of watershed control parameters (see default_watershed_control). Importantly, the sub-model used for lateral water flows - either Francés et al. (2007) or Caviedes-Voullième et al. (2023) - is specified there.
parallelize	Boolean flag to try parallelization (see details).
num_cores	Integer with the number of cores to be used for parallel computation (by default it will use all clusters minus one).
chunk_size	Integer indicating the size of chunks to be sent to different processes (by default, the number of spatial elements divided by the number of cores).
progress	Boolean flag to display progress information for simulations.

Details

See details in spwb_land.

Value

Functions spwb_land_day and spwb_land_day return a sf object:

- geometry: Spatial geometry.
- state: A list of model input objects for each simulated stand.
- aquifer: A numeric vector with the water volume in the aquifer of each cell.
- snowpack: A numeric vector with the snowpack water equivalent volume of each cell.
- result: A list of cell detailed results (only for those indicated in the input), with contents depending on the local model.
- outlet: A logical vector indicating outlet cells.
- MinTemperature: Minimum temperature (degrees Celsius).
- MaxTemperature: Maximum temperature (degrees Celsius).
- PET: Potential evapotranspiration (in mm).
- Rain: Rainfall (in mm).
- Snow: Snowfall (in mm).
- Snowmelt: Snow melt (in mm).
- Interception: Rainfall interception (in mm).
- NetRain: Net rainfall, i.e. throughfall, (in mm).

- `Infiltration`: The amount of water infiltrating into the soil (in mm).
- `InfiltrationExcess`: The amount of water exceeding the soil infiltration capacity (in mm).
- `SaturationExcess`: The amount of water that reaches the soil surface because of soil saturation (in mm).
- `Runoff`: The amount of water exported via surface runoff (in mm).
- `DeepDrainage`: The amount of water draining from soil to the aquifer via deep drainage (in mm).
- `CapillarityRise`: Water entering the soil via capillarity rise (mm) from the water table.
- `SoilEvaporation`: Bare soil evaporation (in mm).
- `Transpiration`: Woody plant transpiration (in mm).
- `HerbTranspiration`: Herbaceous transpiration (in mm).
- `InterflowInput`: The amount of water that reaches the soil of the cell from adjacent cells via subsurface flow (in mm).
- `InterflowOutput`: The amount of water that leaves the soil of the cell towards adjacent cells via subsurface flow (in mm).
- `InterflowBalance`: The balance of water circulating via subsurface flow (in mm).
- `BaseflowInput`: The amount of water that reaches the aquifer of the cell from adjacent cells via groundwater flow (in mm).
- `BaseflowOutput`: The amount of water that leaves the aquifer of the cell towards adjacent cells via groundwater flow (in mm).
- `BaseflowBalance`: The balance of water circulating via groundwater flow (in mm).
- `AquiferExfiltration`: The amount of water of the cell that generates surface runoff due to the aquifer reaching the soil surface (in mm).

Author(s)

Miquel De Cáceres Ainsa, CREAF.

Maria González-Sanchís, Universitat Politecnica de Valencia.

Daniel Caviedes-Voullième, Forschungszentrum Julich.

Mario Morales-Hernández, Universidad de Zaragoza.

References

Francés, F., Vélez, J.I. & Vélez, J.J. (2007). Split-parameter structure for the automatic calibration of distributed hydrological models. Journal of Hydrology, 332, 226–240.

Caviedes-Voullième, D., Morales-Hernández, M., Norman, M.R. & Ogzen-Xian, I. (2023). SERGHEI (SERGHEI-SWE) v1.0: a performance-portable high-performance parallel-computing shallow-water solver for hydrology and environmental hydraulics. Geoscientific Model Development, 16, 977-1008.

See Also

`default_watershed_control`, `spwb_day`, `growth_day`, `spwb_land`,

Examples

```
# Load example watershed data after burnin period
data("example_watershed_burnin")

# Set request for daily model results in cells number 3, 6 (outlet) and 9
example_watershed_burnin$result_cell <- FALSE
example_watershed_burnin$result_cell[c(3,6,9)] <- TRUE

# Get bounding box to determine limits
b <- sf::st_bbox(example_watershed_burnin)
b

# Define a raster topology, using terra package,
# with the same CRS as the watershed. In this example cells have 100 m side.
# Coordinates in the 'sf' object are assumed to be cell centers
r <-terra::rast(xmin = 401380, ymin = 4671820, xmax = 402880, ymax = 4672620,
                nrow = 8, ncol = 15, crs = "epsg:32631")

# Load example meteo data frame from package meteoland
data("examplemeteo")

# Load default medfate parameters
data("SpParamsMED")

# Watershed control parameters (TETIS model; Frances et al. 2007)
ws_control <- default_watershed_control("tetis")

# Launch simulation
date <- "2001-03-01"
sf_out <- spwb_land_day(r, example_watershed_burnin, SpParamsMED, examplemeteo,
                        date = date,
                        watershed_control = ws_control)
```

---

spwb_spatial *Simulations for spatially-distributed forest stands*

Description

Functions that allow calling local models spwb, growth or fordyn, for a set of forest stands distributed in specific locations. No spatial processes are simulated.

Usage

```
spwb_spatial(
  sf,
  SpParams,
  meteo = NULL,
  local_control = defaultControl(),
```

```
  dates = NULL,
  CO2ByYear = numeric(0),
  keep_results = TRUE,
  summary_function = NULL,
  summary_arguments = NULL,
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE,
  local_verbose = FALSE
)

growth_spatial(
  sf,
  SpParams,
  meteo = NULL,
  local_control = defaultControl(),
  dates = NULL,
  CO2ByYear = numeric(0),
  fire_regime = NULL,
  keep_results = TRUE,
  summary_function = NULL,
  summary_arguments = NULL,
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE,
  local_verbose = FALSE
)

fordyn_spatial(
  sf,
  SpParams,
  meteo = NULL,
  local_control = defaultControl(),
  dates = NULL,
  CO2ByYear = numeric(0),
  fire_regime = NULL,
  keep_results = TRUE,
  management_function = NULL,
  summary_function = NULL,
  summary_arguments = NULL,
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE,
  local_verbose = FALSE
)
```

Arguments

sf	An object of class sf with the following columns: • geometry: Spatial geometry. • id: Stand identifiers. • elevation: Elevation above sea level (in m). • slope: Slope (in degrees). • aspect: Aspect (in degrees). • land_cover_type: Land cover type of each grid cell (values should be 'wildland' or 'agriculture'). • forest: Objects of class forest. • soil: Objects of class soil or data frames of physical properties. • state: Objects of class spwbInput or growthInput (optional). • meteo: Data frames with weather data (required if parameter meteo = NULL). • crop_factor: Crop evapo-transpiration factor. Only required for 'agriculture' land cover type. • local_control: A list of control parameters (optional). Used to override function parameter local_control for specific locations (values can be NULL for the remaining ones). • management_arguments: Lists with management arguments. Optional, relevant for fordyn_spatial only. • represented_area_ha: Area represented by each stand in hectares. Optional, relevant for fordyn_spatial when fire_regime is supplied only). • ignition_weights: Relative weights to determine stands to be burned. Optional, relevant for fordyn_spatial when fire_regime is supplied only).
SpParams	A data frame with species parameters (see SpParamsMED).
meteo	Input meteorological data (see section details). If NULL, the function will expect a column 'meteo' in parameter y.
local_control	A list of control parameters (see defaultControl) for function spwb_day or growth_day.
dates	A Date object describing the days of the period to be modeled.
CO2ByYear	A named numeric vector with years as names and atmospheric CO2 concentration (in ppm) as values. Used to specify annual changes in CO2 concentration along the simulation (as an alternative to specifying daily values in meteo).
keep_results	Boolean flag to indicate that point/cell simulation results are to be returned (set to FALSE and use summary functions for large data sets).
summary_function	An appropriate function to calculate summaries (e.g., summary.spwb).
summary_arguments	List with additional arguments for the summary function.
parallelize	Boolean flag to try parallelization (will use all clusters minus one).
num_cores	Integer with the number of cores to be used for parallel computation.

`chunk_size` Integer indicating the size of chuncks to be sent to different processes (by default, the number of spatial elements divided by the number of cores).

`progress` Boolean flag to display progress information of simulations.

`local_verbose` Boolean flag to display detailed progress information in local simulations.

`fire_regime` A list of parameters defining the fire regime (see `create_fire_regime`) or a matrix representing a fire regime instance (see `fire_regime_instance`), to be used in simulations with `fordyn_spatial`. If NULL, wildfires are not simulated.

`management_function` A function that implements forest management actions (see `fordyn`). of such lists, one per spatial unit.

Details

Simulation functions accept different formats for meteorological input (parameter `meteo`). The user may supply two kinds of daily weather sources:

1. A data frame with meteorological data common for all spatial location (spatial variation of weather not considered).
2. An object or (a list of objects) of class `stars` with reference interpolation data created by package `meteoland`. If a list of such *interpolator* objects is supplied, the simulation functions will interpolate on the target locations for the periods covered by each interpolator, but the user will be responsible for supplying interpolators in the correct temporal order.

Alternatively, the user may leave parameter `meteo = NULL` and specify a weather data frame for each element of `y` in a column named 'meteo'.

Fire regimes are only allowed for function `fordyn_spatial`. If an object of class `fire_regime` is supplied, the function will call `fire_regime_instance` to generate a realization of the fire regime before conducting simulations. Alternatively, users can directly supply a fire regime instance matrix, derived from another source (e.g. a fire landscape model). Note that operating with fire regimes assumes all forest stands share the same period of simulation, but enforcing this is left to the user.

Value

An object of class 'sf' containing four elements:

- `geometry`: Spatial geometry.
- `id`: Stand id, taken from the input.
- `state`: A list of `spwbInput` or `growthInput` objects for each simulated stand, to be used in subsequent simulations (see `update_landscape`) or with NULL values whenever simulation errors occurred.
- `forest`: A list of `forest` objects for each simulated stand (only in function `fordyn_spatial`), to be used in subsequent simulations (see `update_landscape`) or with NULL values whenever simulation errors occurred.
- `management_arguments`: A list of management arguments for each simulated stand (only in function `fordyn_spatial` if management function was supplied), to be used in subsequent simulations (see `update_landscape`).

- result: A list of model output for each simulated stand. Some elements can contain an error condition if the simulation resulted in an error. Values will be NULL (or errors) if keep_results = FALSE.
- summary: A list of model output summaries for each simulated stand (if summary_function was not NULL), with NULL values whenever simulation errors occurred.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

spwb, growth, fordyn, spwb_spatial_day, simulation_summary , plot_summary, initialize_landscape, update_landscape

Examples

```
# Load example landscape data
data("example_ifn")

# Load example meteo data frame from package meteoland
data("examplemeteo")

# Load default medfate parameters
data("SpParamsMED")

# Subset two plots to speed-up calculations
example_subset <- example_ifn[31:32, ]

# Perform simulation
dates <- seq(as.Date("2001-03-01"), as.Date("2001-03-15"), by="day")
res <- spwb_spatial(example_subset, SpParamsMED, examplemeteo, dates = dates)

# Perform fordyn simulation for one year (one stand) without management
res_noman <- fordyn_spatial(example_subset, SpParamsMED, examplemeteo)
```

---

spwb_spatial_day *One-day simulation for spatially-distributed forest stands*

Description

Functions that allow calling local models spwb_day or growth_day, for a set of forest stands distributed in specific locations and a given date. No spatial processes are simulated.

Usage

```
spwb_spatial_day(
  sf,
  meteo = NULL,
  date,
  SpParams,
  local_control = defaultControl(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)

growth_spatial_day(
  sf,
  meteo = NULL,
  date,
  SpParams,
  local_control = defaultControl(),
  parallelize = FALSE,
  num_cores = detectCores() - 1,
  chunk_size = NULL,
  progress = TRUE
)
```

Arguments

sf	An object of class sf with landscape information (see spwb_spatial).
meteo	Meteorology data (see spwb_spatial).
date	A string with the date to be simulated.
SpParams	A data frame with species parameters (see SpParamsMED).
local_control	A list of local model control parameters (see defaultControl).
parallelize	Boolean flag to try parallelization (will use all clusters minus one).
num_cores	Integer with the number of cores to be used for parallel computation.
chunk_size	Integer indicating the size of chunks to be sent to different processes (by default, the number of spatial elements divided by the number of cores).
progress	Boolean flag to display progress information for simulations.

Details

Simulation functions accept different formats for meteorological input (described in spwb_spatial).

Value

An object of class sf the same name as the function called containing three elements:

- geometry: Spatial geometry.

- `id`: Stand id, taken from the input.
- `state`: A list of model input objects for each simulated stand, to be used in subsequent simulations.
- `result`: A list of model output for each simulated stand.

Author(s)

Miquel De Cáceres Ainsa, CREAF

See Also

`spwb_day`, `growth_day`, `spwb_spatial`

Examples

```
#Load example landscape data
data("example_ifn")

#Load example meteo data frame from package meteoland
data("examplemeteo")

#Load default medfate parameters
data("SpParamsMED")

#Perform simulation
date <- "2001-03-01"
res <- spwb_spatial_day(example_ifn, examplemeteo, date, SpParamsMED)
```

---

update_landscape *Updates the state of a landscape object*

Description

Updates the state of a spatial object 'x' according to the final state in simulation outcome 'y'

Usage

```
update_landscape(x, y)
```

Arguments

x	An object of class `sf` with the corresponding landscape columns.
y	The object resulting of a simulation previously carried on `x` using `spwb_spatial`, `growth_spatial`, `spwb_land`, etc.

Value

An object of class `sf` with modified state variables.

Author(s)

Miquel De Cáceres Ainsa, CREAF.

See Also

spwb_spatial, spwb_spatial_day, spwb_land

Index