4.1 - Landscape- and regional-scale simulations (practice)

Miquel De Cáceres, Victor Granda, Aitor Ameztegui

Ecosystem Modelling Facility

2022-06-16

Outline

1. Forest and soil initialisation over large areas

2. Parameter estimation for multiple species

3. Tools for model analysis

4. Spatial variation of climate forcing: meteoland

5. Simulation over landscapes: medfateland

1. Forest and soil initialisation over large areas

Data sources

The following is a list of data sources that I commonly use for large-scale initialisation of `forest` and `soil` objects:

Data source	Information	Spatial structure
DEM	Topography	Polygons/raster
Forest maps	Forest composition (dominant species)	Polygons
LiDAR data	Vegetation height	Raster
National forest inventories	Composition and structure on point locations	Points
SoilGrids	Soil texture, bulk density, organic matter,…	Raster
Shagguan et al. (2017)	Soil depth	Raster

1. Forest and soil initialisation over large areas

Data sources

The following is a list of data sources that I commonly use for large-scale initialisation of `forest` and `soil` objects:

Data source	Information	Spatial structure
DEM	Topography	Polygons/raster
Forest maps	Forest composition (dominant species)	Polygons
LiDAR data	Vegetation height	Raster
National forest inventories	Composition and structure on point locations	Points
SoilGrids	Soil texture, bulk density, organic matter,…	Raster
Shagguan et al. (2017)	Soil depth	Raster

You can check on other data sources at EMF website.

1. Forest and soil initialisation over large areas

Initialisation tips

Forest structure and composition

Initialisation of `forest` objects over a **grid** requires defining *imputation procedures* and using multiple data sources:

- Forest inventory data
- Forest maps
- Lidar data.

1. Forest and soil initialisation over large areas

Initialisation tips

Forest structure and composition

Initialisation of `forest` objects over a **grid** requires defining *imputation procedures* and using multiple data sources:

- Forest inventory data
- Forest maps
- Lidar data.

Problem: Most countries lack detailed soil maps but soil properties change substantially at small scales.

1. Forest and soil initialisation over large areas

Initialisation tips

Forest structure and composition

Initialisation of `forest` objects over a **grid** requires defining *imputation procedures* and using multiple data sources:

- Forest inventory data
- Forest maps
- Lidar data.

Problem: Most countries lack detailed soil maps but soil properties change substantially at small scales.

Soils

Relying on *SoilGrids* implies accepting a high degree of uncertainty for some variables.

1. Forest and soil initialisation over large areas

Initialisation tips

Forest structure and composition

Initialisation of `forest` objects over a **grid** requires defining *imputation procedures* and using multiple data sources:

- Forest inventory data
- Forest maps
- Lidar data.

Problem: Most countries lack detailed soil maps but soil properties change substantially at small scales.

Soils

Relying on *SoilGrids* implies accepting a high degree of uncertainty for some variables.

Initialisation of soils requires at least combining *SoilGrids* with additional information on soil depth and rock content.

1. Forest and soil initialisation over large areas

Initialisation tips

Forest structure and composition

Initialisation of `forest` objects over a **grid** requires defining *imputation procedures* and using multiple data sources:

- Forest inventory data
- Forest maps
- Lidar data.

Problem: Most countries lack detailed soil maps but soil properties change substantially at small scales.

Soils

Relying on *SoilGrids* implies accepting a high degree of uncertainty for some variables.

Initialisation of soils requires at least combining *SoilGrids* with additional information on soil depth and rock content.

Surface rock content can serve as a proxy of belowground rock content, but with a high degree of uncertainty!

2. Parameter estimation for multiple species

Creating species parameter tables

Estimating species parameters is the hardest task and most important limitation to the use of process-based models, so *you are not expected to do this by yourself*!

2. Parameter estimation for multiple species

Creating species parameter tables

Estimating species parameters is the hardest task and most important limitation to the use of process-based models, so *you are not expected to do this by yourself*!

General procedure

1. Decide taxonomic treatment according to:
    - Taxonomic resolution of forest data sources (e.g. forest inventory data)
    - Availability of trait data

2. Parameter estimation for multiple species

Creating species parameter tables

Estimating species parameters is the hardest task and most important limitation to the use of process-based models, so *you are not expected to do this by yourself*!

General procedure

1. Decide taxonomic treatment according to:
    - Taxonomic resolution of forest data sources (e.g. forest inventory data)
    - Availability of trait data
2. Store original source codes to be lumped into the same taxon/group (e.g. genus level)

2. Parameter estimation for multiple species

Creating species parameter tables

Estimating species parameters is the hardest task and most important limitation to the use of process-based models, so *you are not expected to do this by yourself*!

General procedure

1. Decide taxonomic treatment according to:
    - Taxonomic resolution of forest data sources (e.g. forest inventory data)
    - Availability of trait data
2. Store original source codes to be lumped into the same taxon/group (e.g. genus level)
3. Initialize medfate's *species* parameter table:

```
SpParams <-  medfateutils::initSpParams()
```

2. Parameter estimation for multiple species

Creating species parameter tables

Estimating species parameters is the hardest task and most important limitation to the use of process-based models, so *you are not expected to do this by yourself*!

General procedure

1. Decide taxonomic treatment according to:
    - Taxonomic resolution of forest data sources (e.g. forest inventory data)
    - Availability of trait data
2. Store original source codes to be lumped into the same taxon/group (e.g. genus level)
3. Initialize medfate's *species* parameter table:

```
SpParams <-  medfateutils::initSpParams()
```

4. Fill species parameter table from multiple sources (different functions in **medfateutils**)

2. Parameter estimation for multiple species

Estimation from forest inventory data

Growth form (tree or shrub) depending on how the species is sampled in the forest source data:

- Trees - Diameter, height,…
- Shrub - Cover, mean height

2. Parameter estimation for multiple species

Estimation from forest inventory data

Growth form (tree or shrub) depending on how the species is sampled in the forest source data:

- Trees - Diameter, height,...
- Shrub - Cover, mean height

The following information can be extracted from forest inventory data:

- Maximum height
- Diameter to height ratio
- Growth rates
- Mortality rates
- Allometric relationships

2. Parameter estimation for multiple species

Estimation from plant trait databases

Source	Database name	Parameters
Asse et al. (2020)		Leaf phenology
Bartlett et al. (2012)		Leaf pressure-volume curve, turgor loss point
Burriel et al. (2004)		Tree allometries
Choat et al. (2012)		Xylem vulnerability
De Cáceres et al. (2019)		Shrub allometries
Delpierre et al. (2019)		Leaf phenology
Duursma et al. (2018)		Minimum stomatal conductance
Hoshika et al. (2018)		Maximum stomatal conductance
Kattge et al. (2020)	TRY	**Multiple traits**
Martin-StPaul et al. (2017)		Turgor loss point
Morris et al. (2016)		Conduit sapwood fraction
Sanchez-Martinez et al. (2020)	HIDRATRY	SLA, wood density, Huber value, xylem efficiency, xylem vulnerability
Tavşanoğlu & Pausas (2006)	BROT2	Life form, leaf duration, SLA, wood density
Vitasse et al. (2011)		Leaf phenology
Yebra et al. (2019)	Globe-LFMC	Fuel moisture content
Zanne et al. (2009)	Global Wood Density Database	Wood density

3. Tools for model analysis

Multiple runs

Model analysis (i.e. calibration, sensitivity analysis, uncertainty analysis, …) involve running a **large number simulations** with different parameter sets.

3. Tools for model analysis

Multiple runs

Model analysis (i.e. calibration, sensitivity analysis, uncertainty analysis, …) involve running a **large number simulations** with different parameter sets.

If we build a matrix `parMatrix` of parameter combinations in rows, we can use function `multiple_runs()` to evaluate them, e.g.:

```
multiple_runs(parMatrix, x1, examplemeteo, latitude = 42, elevation = 100,
              summary_function = sf)
```

where `x1` is the initial model input object and `sf` is a summary function.

3. Tools for model analysis

Multiple runs

Model analysis (i.e. calibration, sensitivity analysis, uncertainty analysis, …) involve running a **large number simulations** with different parameter sets.

If we build a matrix `parMatrix` of parameter combinations in rows, we can use function `multiple_runs()` to evaluate them, e.g.:

```
multiple_runs(parMatrix, x1, examplemeteo, latitude = 42, elevation = 100,
              summary_function = sf)
```

where `x1` is the initial model input object and `sf` is a summary function.

For each parameter combination, `multiple_runs()` will:

1. Modify the values of the target parameter in `x1`
2. Call the simulation function: `spwb()` or `growth()`
3. Call the summary function to extract the desired output.

3. Tools for model analysis

Function factories

Sensitivity analyses and *calibration procedures* often require defining a *function* that accepts values of the parameters to be calibrated and return model outputs or evaluation metrics, e.g.

$$y = g(x_1, x_2, \ldots, x_r)$$

where $x_1, x_2, \ldots, x_r$ is the set of parameter values and y is a scalar corresponding model prediction (e.g. annual transpiration) or an evaluation metric (e.g. mean absolute error of basal area increment).

3. Tools for model analysis

Function factories

Sensitivity analyses and *calibration procedures* often require defining a *function* that accepts values of the parameters to be calibrated and return model outputs or evaluation metrics, e.g.

$$y = g(x_1, x_2, \ldots, x_r)$$

where $x_1, x_2, \ldots, x_r$ is the set of parameter values and y is a scalar corresponding model prediction (e.g. annual transpiration) or an evaluation metric (e.g. mean absolute error of basal area increment).

Package medfate includes *function factories*, i.e. functions that return functions to be used in those calculations.

Function factory	Multiple cohorts	Function returns
`optimization_function()`	No	The scalar of a simulation summary
`optimization_evaluation_function()`	No	The scalar of a simulation evaluation
`optimization_multicohort_function()`	Yes	The scalar of a simulation summary
`optimization_evaluation_multicohort_function()`	Yes	The scalar of a simulation summary

3. Tools for model analysis

Function factories

An example of using the function factory is:

```
of_transp<-optimization_function(parNames = parNames,
                                 x = x1,
                                 meteo = examplemeteo,
                                 latitude = 42, elevation = 100,
                                 summary_function = sf)
```

where `parNames` specifies the target parameters.

3. Tools for model analysis

Function factories

An example of using the function factory is:

```
of_transp<-optimization_function(parNames = parNames,
                                 x = x1,
                                 meteo = examplemeteo,
                                 latitude = 42, elevation = 100,
                                 summary_function = sf)
```

where `parNames` specifies the target parameters.

The object `of_transp` is now our function $y = g(x_1, x_2, \ldots, x_r)$, and we can call it with any parameter combination.

3. Tools for model analysis

Function factories

An example of using the function factory is:

```
of_transp<-optimization_function(parNames = parNames,
                                 x = x1,
                                 meteo = examplemeteo,
                                 latitude = 42, elevation = 100,
                                 summary_function = sf)
```

where `parNames` specifies the target parameters.

The object `of_transp` is now our function $y = g(x_1, x_2, \ldots, x_r)$, and we can call it with any parameter combination.

There is a package vignette illustrating the use of the function factories in different contexts.

Analysis	R package(s)
Global sensitivity analysis	**sensitivity**
Point calibration	**ga** (genetic algorithms), **stats** (gradient search)
Bayesian calibration	**BayesianTools**

4. Spatial variation of climate forcing: meteoland

Purpose

With the aim to assist research of climatic impacts on forests, package **meteoland** provides utilities to estimate daily weather variables at any position over complex terrains:

1. Spatial interpolation of daily weather records from meteorological stations.

4. Spatial variation of climate forcing: meteoland

Purpose

With the aim to assist research of climatic impacts on forests, package **meteoland** provides utilities to estimate daily weather variables at any position over complex terrains:

1. Spatial interpolation of daily weather records from meteorological stations.

2. Statistical correction of meteorological data series (e.g. from climate models).

4. Spatial variation of climate forcing: meteoland

Purpose

With the aim to assist research of climatic impacts on forests, package **meteoland** provides utilities to estimate daily weather variables at any position over complex terrains:

1. Spatial interpolation of daily weather records from meteorological stations.
2. Statistical correction of meteorological data series (e.g. from climate models).
3. Multisite and multivariate stochastic weather generation (underdeveloped).

4. Spatial variation of climate forcing: meteoland

Purpose

With the aim to assist research of climatic impacts on forests, package **meteoland** provides utilities to estimate daily weather variables at any position over complex terrains:

1. Spatial interpolation of daily weather records from meteorological stations.
2. Statistical correction of meteorological data series (e.g. from climate models).
3. Multisite and multivariate stochastic weather generation (underdeveloped).

Installation

From **CRAN** (stable versions; now ver. **1.0.2**):

```
install.packages("meteoland")
```

4. Spatial variation of climate forcing: meteoland

Purpose

With the aim to assist research of climatic impacts on forests, package **meteoland** provides utilities to estimate daily weather variables at any position over complex terrains:

1. Spatial interpolation of daily weather records from meteorological stations.
2. Statistical correction of meteorological data series (e.g. from climate models).
3. Multisite and multivariate stochastic weather generation (underdeveloped).

Installation

From **CRAN** (stable versions; now ver. **1.0.2**):

```
install.packages("meteoland")
```

From **GitHub** (now ver. **1.0.3**):

```
remotes::install_github("emf-creaf/meteoland")
```

4. Spatial variation of climate forcing: meteoland

Spatial topography classes

Three classes are defined to represent the variation of topographic features (i.e., elevation, slope and aspect) over space, extending S4 classes of package **sp**:

- Class **`SpatialPointsTopography`** extends `SpatialPointsDataFrame` and represents the topographic features of a set of points in a landscape.
- Class **`SpatialGridTopography`** extends `SpatialGridDataFrame` and represents the continuous variation of topographic features over a full spatial grid.
- Class **`SpatialPixelsTopography`** extends `SpatialPixelsDataFrame` and represents the continuous variation of topographic features over a set if cells in a grid.

4. Spatial variation of climate forcing: meteoland

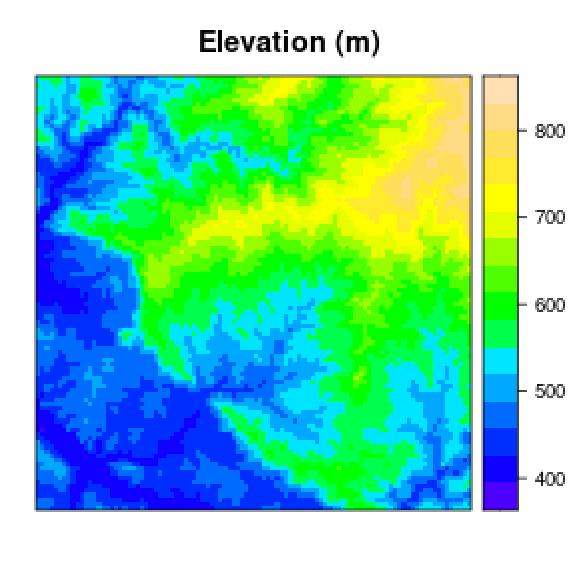
Spatial topography classes

Three classes are defined to represent the variation of topographic features (i.e., elevation, slope and aspect) over space, extending S4 classes of package **sp**:

- Class **`SpatialPointsTopography`** extends `SpatialPointsDataFrame` and represents the topographic features of a set of points in a landscape.
- Class **`SpatialGridTopography`** extends `SpatialGridDataFrame` and represents the continuous variation of topographic features over a full spatial grid.
- Class **`SpatialPixelsTopography`** extends `SpatialPixelsDataFrame` and represents the continuous variation of topographic features over a set if cells in a grid.

Data frames in topography classes have only three attributes:

- `elevation` in meters a.s.l.
- `slope` in degrees from the horizontal plane.
- `aspect` in degrees from North.



4. Spatial variation of climate forcing: meteoland

Spatial meteorology classes

Analogously to topography classes, three spatial classes are used to represent the variation of daily meteorology over space, also extending classes in **sp**:

- Class **`SpatialPointsMeteorology`** extends `SpatialPoints` and represents daily meteorology series for a set of points in a landscape.
- Class **`SpatialGridMeteorology`** extends `SpatialGrid` and represents the continuous variation of daily meteorology across a grid of cells.
- Class **`SpatialPixelsMeteorology`** extends `SpatialPixels` and represents the variation of daily meteorology for a set of pixels (cells) of a spatial grid.

4. Spatial variation of climate forcing: meteoland

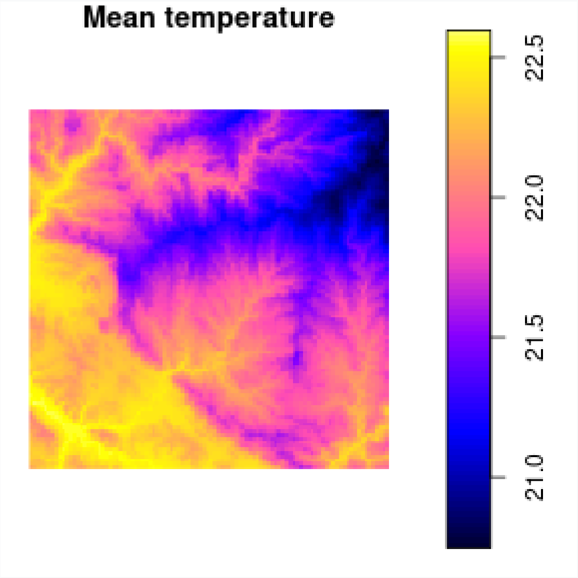
Spatial meteorology classes

Analogously to topography classes, three spatial classes are used to represent the variation of daily meteorology over space, also extending classes in **sp**:

- Class **`SpatialPointsMeteorology`** extends `SpatialPoints` and represents daily meteorology series for a set of points in a landscape.
- Class **`SpatialGridMeteorology`** extends `SpatialGrid` and represents the continuous variation of daily meteorology across a grid of cells.
- Class **`SpatialPixelsMeteorology`** extends `SpatialPixels` and represents the variation of daily meteorology for a set of pixels (cells) of a spatial grid.

Spatial meteorology classes have two important slots:

- **`dates`** - a vector of days specifying a time period.
- **`data`** - a vector of data frames with the meteorological data.
    - One data frame for each point in `SpatialPointsMeteorology`.
    - One data frame for each day in `SpatialGridMeteorology` and `SpatialPixelsMeteorology`.



4. Spatial variation of climate forcing: meteoland

Weather interpolation

Interpolation methods

The general procedure for interpolation is very similar to the one that underpins the U.S. DAYMET dataset (https://daymet.ornl.gov/).

4. Spatial variation of climate forcing: meteoland

Weather interpolation

Interpolation methods

The general procedure for interpolation is very similar to the one that underpins the U.S. DAYMET dataset (https://daymet.ornl.gov/).

- *Minimum temperature*, *maximum temperature* and *precipitation* are interpolated from a set of point weather records using truncated Gaussian filters, while accounting for the relationship between these variables and elevation (Thornton et al. 1997).

4. Spatial variation of climate forcing: meteoland

Weather interpolation

Interpolation methods

The general procedure for interpolation is very similar to the one that underpins the U.S. DAYMET dataset (https://daymet.ornl.gov/).

- *Minimum temperature*, *maximum temperature* and *precipitation* are interpolated from a set of point weather records using truncated Gaussian filters, while accounting for the relationship between these variables and elevation (Thornton et al. 1997).

- *Relative humidity* can be either interpolated (in fact, dew-point temperature is the variable being interpolated) or predicted from temperature estimates, depending on whether it was measured in in the set of reference points (surface stations).

4. Spatial variation of climate forcing: meteoland

Weather interpolation

Interpolation methods

The general procedure for interpolation is very similar to the one that underpins the U.S. DAYMET dataset (https://daymet.ornl.gov/).

- *Minimum temperature*, *maximum temperature* and *precipitation* are interpolated from a set of point weather records using truncated Gaussian filters, while accounting for the relationship between these variables and elevation (Thornton et al. 1997).

- *Relative humidity* can be either interpolated (in fact, dew-point temperature is the variable being interpolated) or predicted from temperature estimates, depending on whether it was measured in in the set of reference points (surface stations).

- *Potential solar radiation* is estimated taking into account latitude, seasonality, aspect and slope. Actual solar irradiance is then estimated from potential radiation by including the effect of atmosphere transmittance using the predictions of temperature range, relative humidity and precipitation (Thornton & Running 1999).

4. Spatial variation of climate forcing: meteoland

Weather interpolation

Interpolation methods

The general procedure for interpolation is very similar to the one that underpins the U.S. DAYMET dataset (https://daymet.ornl.gov/).

- *Minimum temperature*, *maximum temperature* and *precipitation* are interpolated from a set of point weather records using truncated Gaussian filters, while accounting for the relationship between these variables and elevation (Thornton et al. 1997).

- *Relative humidity* can be either interpolated (in fact, dew-point temperature is the variable being interpolated) or predicted from temperature estimates, depending on whether it was measured in in the set of reference points (surface stations).

- *Potential solar radiation* is estimated taking into account latitude, seasonality, aspect and slope. Actual solar irradiance is then estimated from potential radiation by including the effect of atmosphere transmittance using the predictions of temperature range, relative humidity and precipitation (Thornton & Running 1999).

- The *wind vector* (wind direction and wind speed) is interpolated by using weather station records and static wind fields.

4. Spatial variation of climate forcing: meteoland

Weather interpolation

MeteorologyInterpolationData

Package meteoland stores weather series for reference locations (surface weather stations) and interpolation parameters in a single object of class `MeteorologyInterpolationData`.

4. Spatial variation of climate forcing: meteoland

Weather interpolation

MeteorologyInterpolationData

Package meteoland stores weather series for reference locations (surface weather stations) and interpolation parameters in a single object of class `MeteorologyInterpolationData`.

Warning: Collecting and assembling surface weather records into an object of `MeteorologyInterpolationData` is the tedious part of using package **meteoland**.

4. Spatial variation of climate forcing: meteoland

Weather interpolation

MeteorologyInterpolationData

Package meteoland stores weather series for reference locations (surface weather stations) and interpolation parameters in a single object of class `MeteorologyInterpolationData`.

Warning: Collecting and assembling surface weather records into an object of `MeteorologyInterpolationData` is the tedious part of using package **meteoland**.

Interpolation functions

Interpolation is conducted using different R functions depending on the spatial input:

Spatial input	Interpolation function	Spatial output
`SpatialPointsTopography`	`interpolationpoints()`	`SpatialPointsMeteorology`
`SpatialGridTopography`	`interpolationgrid()`	`SpatialGridMeteorology`
`SpatialPixelsTopography`	`interpolationpixels()`	`SpatialPixelsMeteorology`

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

Concept

The general idea of correction is that a fine-scale weather series is compared to a coarse-scale series for a *reference* (historical) period. The result of this comparison can be used to correct coarse-scale weather series for a *target* (e.g. future) period.

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

Concept

The general idea of correction is that a fine-scale weather series is compared to a coarse-scale series for a *reference* (historical) period. The result of this comparison can be used to correct coarse-scale weather series for a *target* (e.g. future) period.

Correction methods

Let x_i be the value of the variable of the more accurate (e.g. local) series for a given day i and u_i the corresponding value for the less accurate series (e.g., climate model output).

Users can choose between three different types of corrections:

1. *Unbiasing*: consists in subtracting, from the series to be corrected, the average difference between the two series for the reference period: $\theta = \sum_i^n (u_i - x_i)/n$.

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

Concept

The general idea of correction is that a fine-scale weather series is compared to a coarse-scale series for a *reference* (historical) period. The result of this comparison can be used to correct coarse-scale weather series for a *target* (e.g. future) period.

Correction methods

Let x_i be the value of the variable of the more accurate (e.g. local) series for a given day i and u_i the corresponding value for the less accurate series (e.g., climate model output).

Users can choose between three different types of corrections:

1. *Unbiasing*: consists in subtracting, from the series to be corrected, the average difference between the two series for the reference period: $\theta = \sum_i^n (u_i - x_i)/n$.

2. *Scaling*: A slope is calculated by regressing u on x through the origin using data of the reference period. The slope can then be used as scaling factor to multiply the values of u for any day of the period of interest.

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

Concept

The general idea of correction is that a fine-scale weather series is compared to a coarse-scale series for a *reference* (historical) period. The result of this comparison can be used to correct coarse-scale weather series for a *target* (e.g. future) period.

Correction methods

Let x_i be the value of the variable of the more accurate (e.g. local) series for a given day i and u_i the corresponding value for the less accurate series (e.g., climate model output).

Users can choose between three different types of corrections:

1. *Unbiasing*: consists in subtracting, from the series to be corrected, the average difference between the two series for the reference period: $\theta = \sum_i^n (u_i - x_i)/n$.

2. *Scaling*: A slope is calculated by regressing u on x through the origin using data of the reference period. The slope can then be used as scaling factor to multiply the values of u for any day of the period of interest.

3. *Empirical quantile mapping*: Consists in comparing the empirical cumulative distribution function (CDF) of the two series for the reference period, and this mapping is used to correct values of u for the target period.

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

MeteorologyUncorrectedData

Statistical correction needs an object of class `MeteorologyUncorrectedData`, containing the coarse-scale data to be corrected (for both the reference and target periods) and the correction method to be used for each variable. e.g.

```
u <- MeteorologyUncorrectedData(sp, u_reference, u_target, ...)
```

4. Spatial variation of climate forcing: meteoland

Downscaling and bias correction

MeteorologyUncorrectedData

Statistical correction needs an object of class `MeteorologyUncorrectedData`, containing the coarse-scale data to be corrected (for both the reference and target periods) and the correction method to be used for each variable. e.g.

```
u <- MeteorologyUncorrectedData(sp, u_reference, u_target, ...)
```

Correction function

Correction is performed using function `correctionpoints()`, which takes as input the object of class `MeteorologyUncorrectedData` and an object of class `SpatialPointsMeteorology` with the fine-scale data for the reference period.

```
y <- correctionpoints(u, x)
```

The function will take all points in `x` as spatial target locations to perform the correction (and implicitly downscaling) of the coarse-scale data in `u`.

5. Simulation over landscapes: medfateland

Purpose

The R package **medfateland** (under development) has been designed to run simulations of forest functioning and dynamics at the landscape and regional scales.

5. Simulation over landscapes: medfateland

Purpose

The R package **medfateland** (under development) has been designed to run simulations of forest functioning and dynamics at the landscape and regional scales.

The package allows executing the models available in package **medfate** on points and cells within landscape, in either *sequentially* or using *parallel computation*.

5. Simulation over landscapes: medfateland

Purpose

The R package **medfateland** (under development) has been designed to run simulations of forest functioning and dynamics at the landscape and regional scales.

The package allows executing the models available in package **medfate** on points and cells within landscape, in either *sequentially* or using *parallel computation*.

In addition, **medfateland** implements spatial hydrological processes for simulations in forested watersheds.

5. Simulation over landscapes: medfateland

Purpose

The R package **medfateland** (under development) has been designed to run simulations of forest functioning and dynamics at the landscape and regional scales.

The package allows executing the models available in package **medfate** on points and cells within landscape, in either *sequentially* or using *parallel computation*.

In addition, **medfateland** implements spatial hydrological processes for simulations in forested watersheds.

Installation and documentation

The package is available at GitHub only:

```
remotes::install_github("emf-creaf/medfateland")
```

5. Simulation over landscapes: medfateland

Purpose

The R package **medfateland** (under development) has been designed to run simulations of forest functioning and dynamics at the landscape and regional scales.

The package allows executing the models available in package **medfate** on points and cells within landscape, in either *sequentially* or using *parallel computation*.

In addition, **medfateland** implements spatial hydrological processes for simulations in forested watersheds.

Installation and documentation

The package is available at GitHub only:

```
remotes::install_github("emf-creaf/medfateland")
```

Information about the design of medfateland can be found in its website and in medfate's reference book.

5. Simulation over landscapes: medfateland

Data structures

Package **medfateland** offers three *spatial classes* that inherit fields from three corresponding classes in package **meteoland**:

- `SpatialPointsLandscape`: represents a set of forest stands (including soil description) as points within a landscape. Extends class `SpatialPointsTopography`.
- `SpatialPixelsLandscape`: represents a set of forests (including soil description) or other land cover units (i.e. agricultural, rock outcrops or urban areas) as pixels within a gridded landscape. Extends class `SpatialPixelsTopography`.
- `SpatialGridLandscape`: represents a set of forests (including soil description) or other land cover units (i.e. agricultural, rock outcrops or urban areas) as pixels within a complete grid. Extends class `SpatialGridTopography`.

5. Simulation over landscapes: medfateland

Data structures

Package **medfateland** offers three *spatial classes* that inherit fields from three corresponding classes in package **meteoland**:

- `SpatialPointsLandscape`: represents a set of forest stands (including soil description) as points within a landscape. Extends class `SpatialPointsTopography`.
- `SpatialPixelsLandscape`: represents a set of forests (including soil description) or other land cover units (i.e. agricultural, rock outcrops or urban areas) as pixels within a gridded landscape. Extends class `SpatialPixelsTopography`.
- `SpatialGridLandscape`: represents a set of forests (including soil description) or other land cover units (i.e. agricultural, rock outcrops or urban areas) as pixels within a complete grid. Extends class `SpatialGridTopography`.

An additional spatial class is defined for watershed ecohydrological modelling:

- `DistributedWatershed`: Represents a (forested) watershed, including land cover units (i.e. agricultural, rock outcrops or urban areas), forest and soil information as well as bedrock properties. Extends class `SpatialPixelsLandscape`.

5. Simulation over landscapes: medfateland

Data structures

There are example spatial landscape objects in the package, e.g. a `SpatialPointsLandscape`:

```
data("examplepointslandscape")
```

5. Simulation over landscapes: medfateland

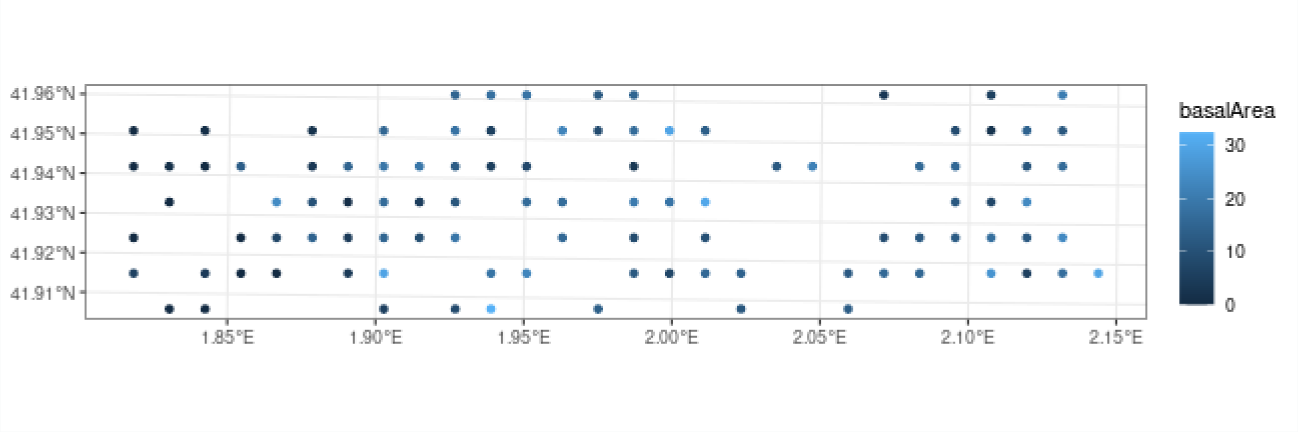
Data structures

There are example spatial landscape objects in the package, e.g. a `SpatialPointsLandscape`:

```
data("examplepointslandscape")
```

Using `plot()` functions for spatial landscape objects, we can draw maps of some variables using:

```
plot(examplepointslandscape, "basalArea")
```

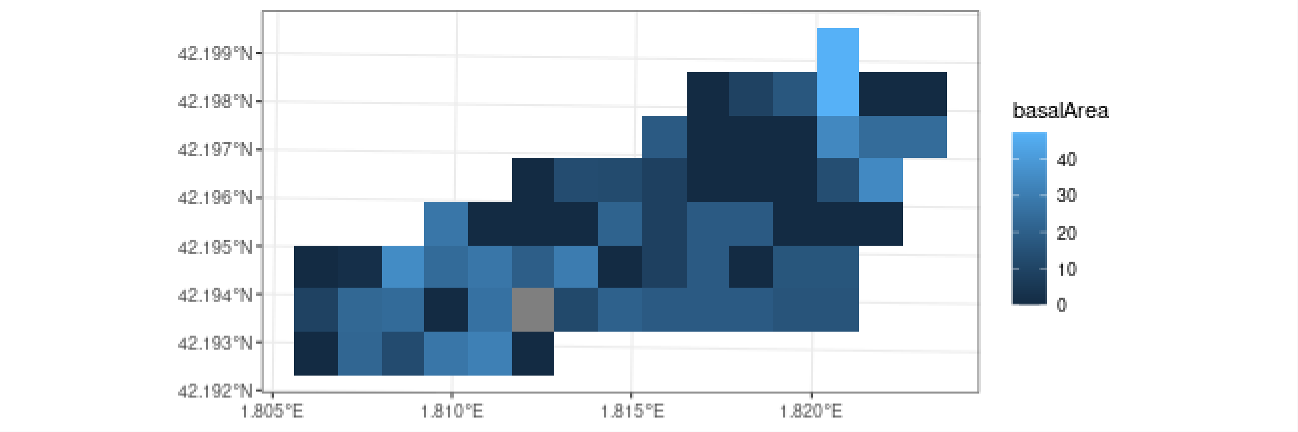


5. Simulation over landscapes: medfateland

Data structures

Another example concerns a `DistributedWatershed`:

```
data("examplewatershed")
plot(examplewatershed, "basalArea")
```

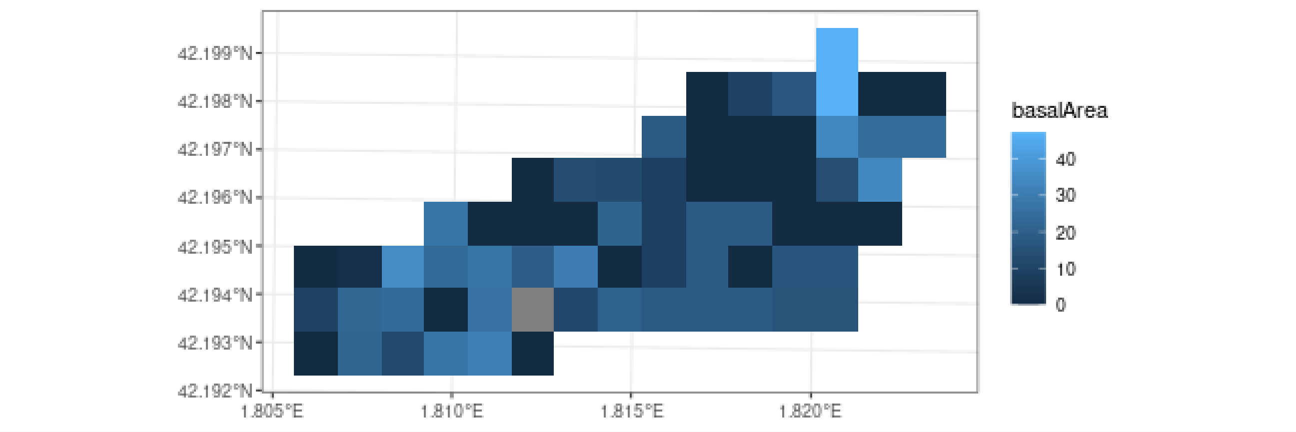


5. Simulation over landscapes: medfateland

Data structures

Another example concerns a `DistributedWatershed`:

```
data("examplewatershed")
plot(examplewatershed, "basalArea")
```



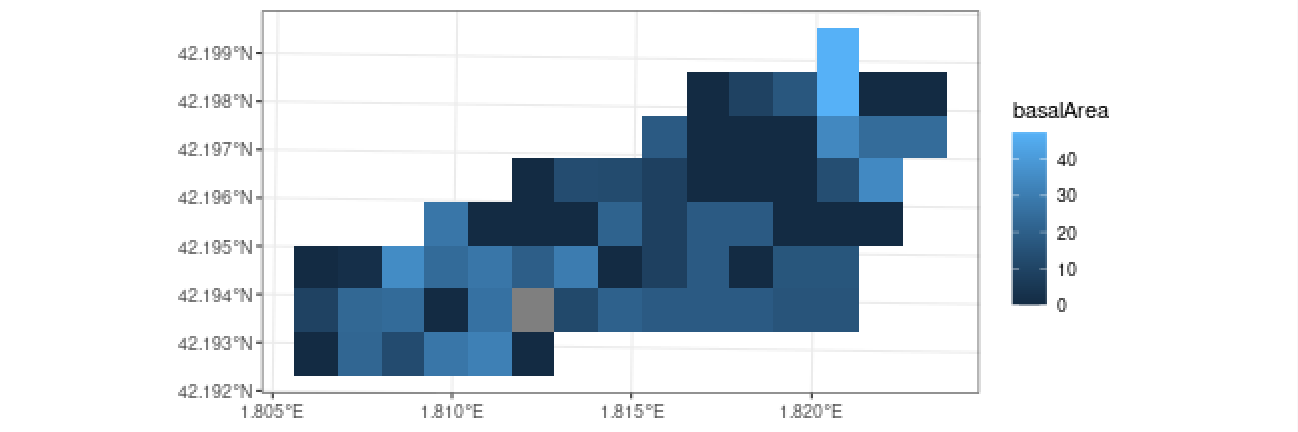
The set of maps available can be known by inspecting the help of function `getLandscapeLayer()`.

5. Simulation over landscapes: medfateland

Data structures

Another example concerns a `DistributedWatershed`:

```
data("examplewatershed")
plot(examplewatershed, "basalArea")
```



The set of maps available can be known by inspecting the help of function `getLandscapeLayer()`.

Alternatively, the package provides function `shinyplotland()` to display maps interactively.

5. Simulation over landscapes: medfateland

Dynamic simulation functions

A large number of simulation functions are included in package **medfateland**:

Spatial structure	Water balance (1 day)	Water balance (n days)	Forest growth	Forest dynamics
`SpatialPointsLandscape`	`spwbpoints_day()`	`spwbpoints()`	`growthpoints()`	`fordynpoints()`
`SpatialPixelsLandscape`	`spwbpixels_day()`	`spwbpixels()`	`growthpixels()`	`fordynpixels()`
`SpatialGridLandscape`	`spwbgrid_day()`	`spwbgrid()`	`growthgrid()`	`fordyngrid()`
`DistributedWatershed`		`spwbland()`	`growthland()`	

5. Simulation over landscapes: medfateland

Dynamic simulation functions

A large number of simulation functions are included in package **medfateland**:

Spatial structure	Water balance (1 day)	Water balance (n days)	Forest growth	Forest dynamics
`SpatialPointsLandscape`	`spwbpoints_day()`	`spwbpoints()`	`growthpoints()`	`fordynpoints()`
`SpatialPixelsLandscape`	`spwbpixels_day()`	`spwbpixels()`	`growthpixels()`	`fordynpixels()`
`SpatialGridLandscape`	`spwbgrid_day()`	`spwbgrid()`	`growthgrid()`	`fordyngrid()`
`DistributedWatershed`		`spwbland()`	`growthland()`	

Most of these functions make internal calls to `spwb()`, `growth()` or `fordyn()` on points or grid cells of the spatial classes.

5. Simulation over landscapes: medfateland

Dynamic simulation functions

A large number of simulation functions are included in package **medfateland**:

Spatial structure	Water balance (1 day)	Water balance (n days)	Forest growth	Forest dynamics
`SpatialPointsLandscape`	`spwbpoints_day()`	`spwbpoints()`	`growthpoints()`	`fordynpoints()`
`SpatialPixelsLandscape`	`spwbpixels_day()`	`spwbpixels()`	`growthpixels()`	`fordynpixels()`
`SpatialGridLandscape`	`spwbgrid_day()`	`spwbgrid()`	`growthgrid()`	`fordyngrid()`
`DistributedWatershed`		`spwbland()`	`growthland()`	

Most of these functions make internal calls to `spwb()`, `growth()` or `fordyn()` on points or grid cells of the spatial classes.

Important: Since most functions do not account for spatial processes, there are parameters to allow the user to specify *parallel computation*.

5. Simulation over landscapes: medfateland

Climate forcing in large-scale simulations

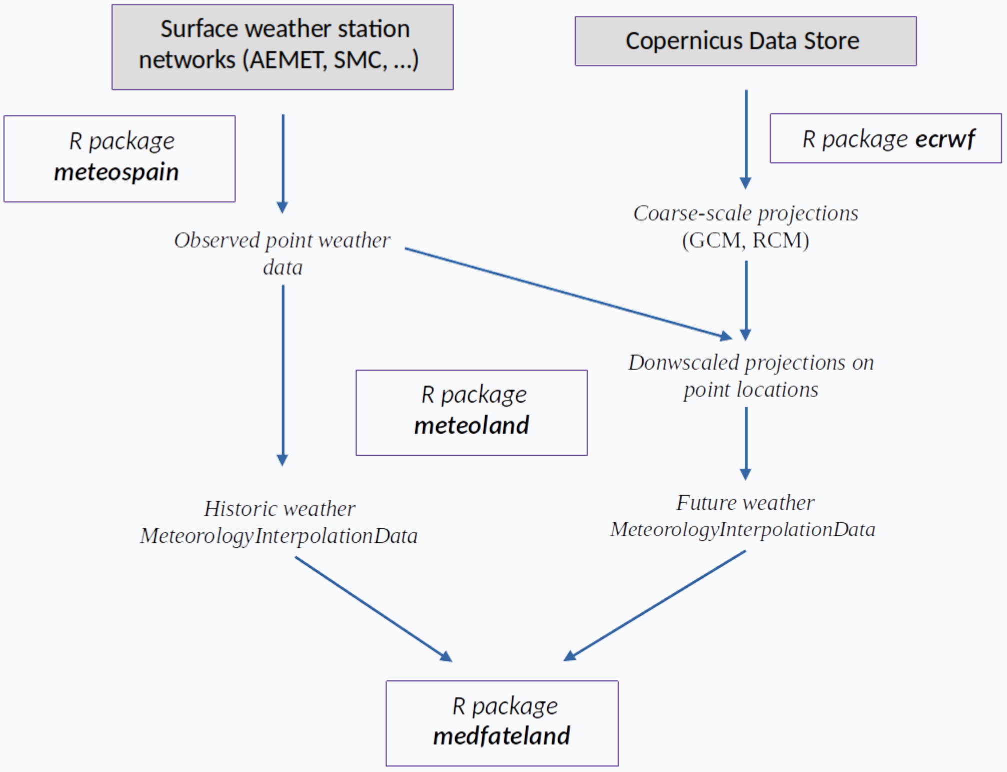
Simulation functions of **medfateland** accept objects of class `MeteorologyInterpolationData` as input, which allows performing interpolation at the time of performing simulations.

5. Simulation over landscapes: medfateland

Climate forcing in large-scale simulations

Simulation functions of **medfateland** accept objects of class `MeteorologyInterpolationData` as input, which allows performing interpolation at the time of performing simulations.

The following workflow can be envisaged for large-scale simulations with **medfateland**:



M.C. Escher - Belvedere, 1958