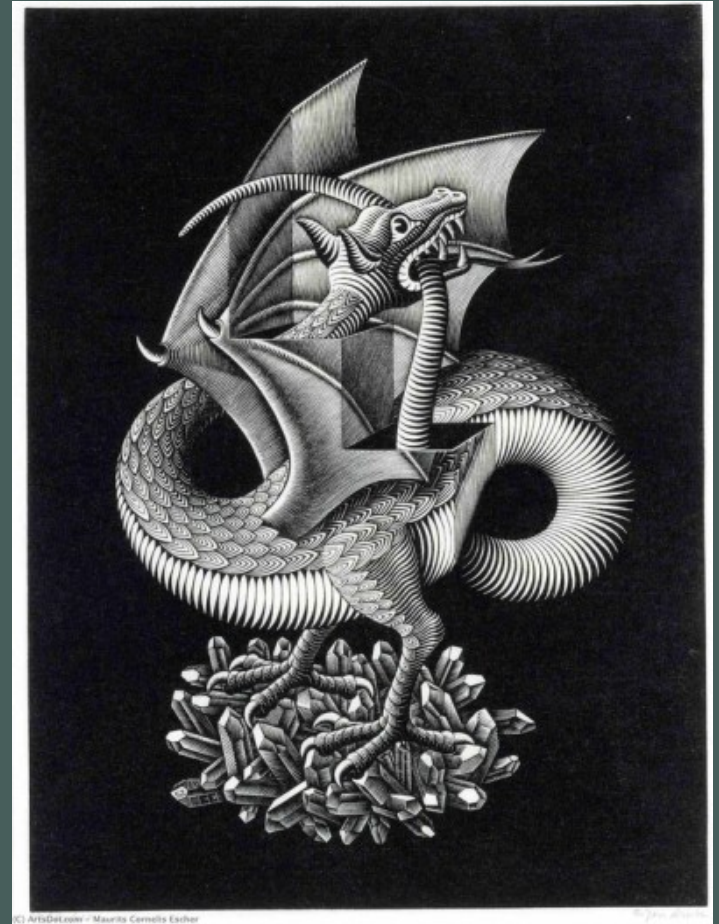


# 1.3 - Model inputs

Miquel De Cáceres, Victor Granda, Aitor Ameztegui

Ecosystem Modelling Facility

2022-06-13



# Outline

1. Species parameters
2. Forest input
3. Vertical profiles
4. Soil input
5. Weather input
6. Simulation control
7. Simulation input object

# 1. Species parameters

Simulation models in **medfate** require a `data.frame` with species parameter values.

# 1. Species parameters

Simulation models in **medfate** require a `data.frame` with species parameter values.

The package includes a default data set of parameter values for 217 Mediterranean taxa.

```
data("SpParamsMED")
```

# 1. Species parameters

Simulation models in **medfate** require a `data.frame` with species parameter values.

The package includes a default data set of parameter values for 217 Mediterranean taxa.

```
data("SpParamsMED")
```

A large number of parameters (122 columns) can be found in `SpParamsMED`, which may be intimidating.

# 1. Species parameters

Simulation models in **medfate** require a `data.frame` with species parameter values.

The package includes a default data set of parameter values for 217 Mediterranean taxa.

```
data("SpParamsMED")
```

A large number of parameters (122 columns) can be found in `SpParamsMED`, which may be intimidating.

You can find parameter definitions in table `SpParamsDefinition`:

```
data("SpParamsDefinition")
```

# 1. Species parameters

The following table shows parameter definitions and units:

Show  entries Search:

	ParameterName	ParameterGroup	Definition	Type	Units
1	Name	Identity	Taxon names (species binomials or genus)	String	
2	IFNcodes	Identity	Codes in the forest inventory, separated by '/'	String	
3	SpIndex	Identity	Species index 0,1,2,♂	Integer	
4	Genus	Taxonomic identity	Taxonomic genus	String	
5	Order	Taxonomic identity	Taxonomical order	String	
6	Family	Taxonomic identity	Taxonomical family	String	

Showing 1 to 6 of 122 entries

Previous  2 3 4 5 ... 21 Next

## 2. Forest input

### Forest class

Each *forest plot* is represented in an object of class `forest`, a list that contains several elements.

```
forest <- exampleforestMED
```



## 2. Forest input

### Forest class

Each *forest plot* is represented in an object of class `forest`, a list that contains several elements.

```
forest <- exampleforestMED
```

The most important items are two data frames, `treeData` (for trees):

```
forest$treeData
```

```
##   Species    N   DBH Height Z50  Z95
## 1     148 168 37.55    800 100  600
## 2     168 384 14.60    660 300 1000
```

## 2. Forest input

### Forest class

Each *forest plot* is represented in an object of class `forest`, a list that contains several elements.

```
forest <- exampleforestMED
```

The most important items are two data frames, `treeData` (for trees):

```
forest$treeData
```

```
##   Species    N   DBH Height Z50  Z95
## 1      148 168 37.55    800 100  600
## 2      168 384 14.60    660 300 1000
```

and `shrubData` (for shrubs):

```
forest$shrubData
```

```
##   Species Cover Height Z50  Z95
## 1      165  3.75    80 200 1000
```

## 2. Forest input

### Forest class

The two data frames share many variables...

## 2. Forest input

### Forest class

The two data frames share many variables...

#### Tree data

Variable	Definition
Species	Species numerical code (should match SpIndex in SpParams)
N	Density of trees (in individuals per hectare)
DBH	Tree diameter at breast height (in cm)
Height	Tree total height (in cm)
Z50	Soil depth corresponding to 50% of fine roots (mm)
Z95	Soil depth corresponding to 95% of fine roots (mm)

## 2. Forest input

### Forest class

The two data frames share many variables...

#### Tree data

Variable	Definition
Species	Species numerical code (should match SpIndex in SpParams)
N	Density of trees (in individuals per hectare)
DBH	Tree diameter at breast height (in cm)
Height	Tree total height (in cm)
Z50	Soil depth corresponding to 50% of fine roots (mm)
Z95	Soil depth corresponding to 95% of fine roots (mm)

#### Shrub data

Variable	Definition
Species	Species numerical code (should match SpIndex in SpParams)
Cover	Shrub cover (%)
Height	Shrub total height (in cm)
Z50	Soil depth corresponding to 50% of fine roots (mm)
Z95	Soil depth corresponding to 95% of fine roots (mm)

## 2. Forest input

### Forest class

The two data frames share many variables...

#### Tree data

Variable	Definition
Species	Species numerical code (should match SpIndex in SpParams)
N	Density of trees (in individuals per hectare)
DBH	Tree diameter at breast height (in cm)
Height	Tree total height (in cm)
Z50	Soil depth corresponding to 50% of fine roots (mm)
Z95	Soil depth corresponding to 95% of fine roots (mm)

#### Shrub data

Variable	Definition
Species	Species numerical code (should match SpIndex in SpParams)
Cover	Shrub cover (%)
Height	Shrub total height (in cm)
Z50	Soil depth corresponding to 50% of fine roots (mm)
Z95	Soil depth corresponding to 95% of fine roots (mm)

**Important:** medfate's *naming conventions* for tree cohorts and shrub cohorts uses T or S, the row number and species numerical code (e.g. "T1\_148" for the first tree cohort, corresponding to *Pinus halepensis*).

## 2. Forest input

### Creating a 'forest' object from forest inventory data

Forest inventories can be conducted in different ways, which means that the starting form of forest data is diverse.

## 2. Forest input

### Creating a 'forest' object from forest inventory data

Forest inventories can be conducted in different ways, which means that the starting form of forest data is diverse.

Building forest objects from inventory data will always require some data wrangling, but package **medfateutils** provides functions that may be helpful:



## 2. Forest input

### Creating a 'forest' object from forest inventory data

Forest inventories can be conducted in different ways, which means that the starting form of forest data is diverse.

Building forest objects from inventory data will always require some data wrangling, but package **medfateutils** provides functions that may be helpful:

Function	Description
<code>forest_mapShrubTable()</code>	Helps filling shrubData table
<code>forest_mapTreeTable()</code>	Helps filling treeData table
<code>forest_mapWoodyTables()</code>	Helps filling a forest object
<code>IFN2forestlist()</code>	Creates a list of forest objects from Spanish forest inventory data

## 2. Forest input

### Forest attributes

The **medfate** package includes a number of functions to examine properties of the plants conforming a forest object:

- `plant_*`: Cohort-level information (species name, id, leaf area index, height...).
- `species_*`: Species-level attributes (e.g. basal area, leaf area index).
- `stand_*`: Stand-level attributes (e.g. basal area).

```
plant_basalArea(forest)
```

```
##      T1_148      T2_168      S1_165  
## 18.604547   6.428755          NA
```

```
stand_basalArea(forest)
```

```
## [1] 25.0333
```

## 2. Forest input

### Forest attributes

The **medfate** package includes a number of functions to examine properties of the plants conforming a forest object:

- `plant_*`: Cohort-level information (species name, id, leaf area index, height...).
- `species_*`: Species-level attributes (e.g. basal area, leaf area index).
- `stand_*`: Stand-level attributes (e.g. basal area).

```
plant_basalArea(forest)
```

```
##      T1_148      T2_168      S1_165
## 18.604547   6.428755         NA
```

```
stand_basalArea(forest)
```

```
## [1] 25.0333
```

```
plant_LAI(forest, SpParamsMED)
```

```
##      T1_148      T2_168      S1_165
## 0.96734365 0.86167321 0.03928201
```

```
stand_LAI(forest, SpParamsMED)
```

```
## [1] 1.868299
```

## 2. Forest input

### Aboveground data

An important information for simulation model is the estimation of initial **leaf area index** and **crown dimensions** for each plant cohort, which is normally done using *allometries*.

## 2. Forest input

### Aboveground data

An important information for simulation model is the estimation of initial **leaf area index** and **crown dimensions** for each plant cohort, which is normally done using *allometries*.

We can illustrate this step using function `forest2aboveground()`:

```
above <- forest2aboveground(forest, SpParamsMED)
above
```

##	SP	N	DBH	Cover	H	CR	LAI_live	LAI_expanded	LAI_dead
## T1_148	148	168.0000	37.55	NA	800	0.6605196	0.96734365	0.96734365	0
## T2_168	168	384.0000	14.60	NA	660	0.6055642	0.86167321	0.86167321	0
## S1_165	165	749.4923	NA	3.75	80	0.8032817	0.03928201	0.03928201	0

where species-specific allometric coefficients are taken from `SpParamsMED`.

## 2. Forest input

### Aboveground data

An important information for simulation model is the estimation of initial **leaf area index** and **crown dimensions** for each plant cohort, which is normally done using *allometries*.

We can illustrate this step using function `forest2aboveground()`:

```
above <- forest2aboveground(forest, SpParamsMED)
above
```

##	SP	N	DBH	Cover	H	CR	LAI_live	LAI_expanded	LAI_dead
## T1_148	148	168.0000	37.55	NA	800	0.6605196	0.96734365	0.96734365	0
## T2_168	168	384.0000	14.60	NA	660	0.6055642	0.86167321	0.86167321	0
## S1_165	165	749.4923	NA	3.75	80	0.8032817	0.03928201	0.03928201	0

where species-specific allometric coefficients are taken from `SpParamsMED`.

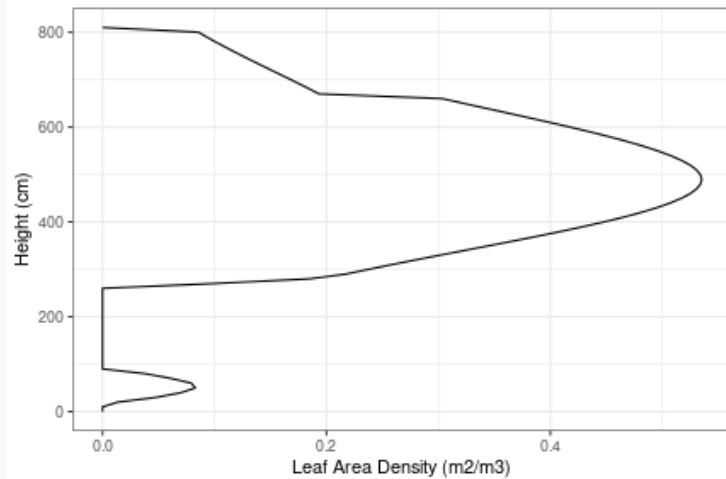
Users will not normally call `forest2aboveground()`, but is important to understand what is going on behind the scenes.

# 3. Vertical profiles

## Leaf distribution

Vertical leaf area distribution (at the cohort-, species- or stand-level) can be examined using:

```
vprofile_leafAreaDensity(forest, SpParamsMED)
```

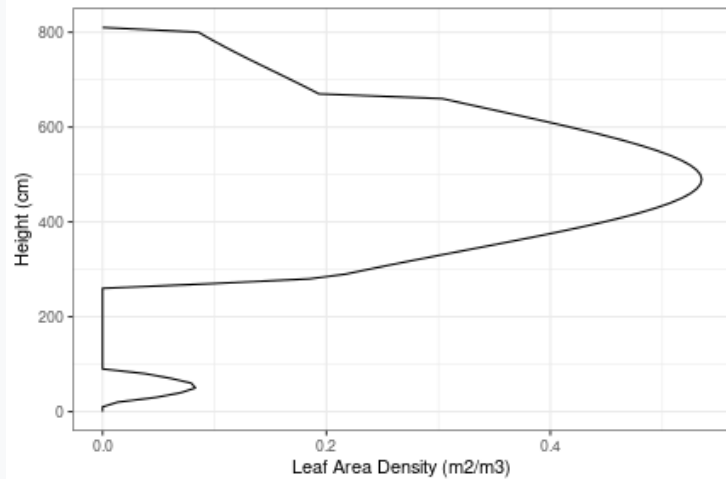


### 3. Vertical profiles

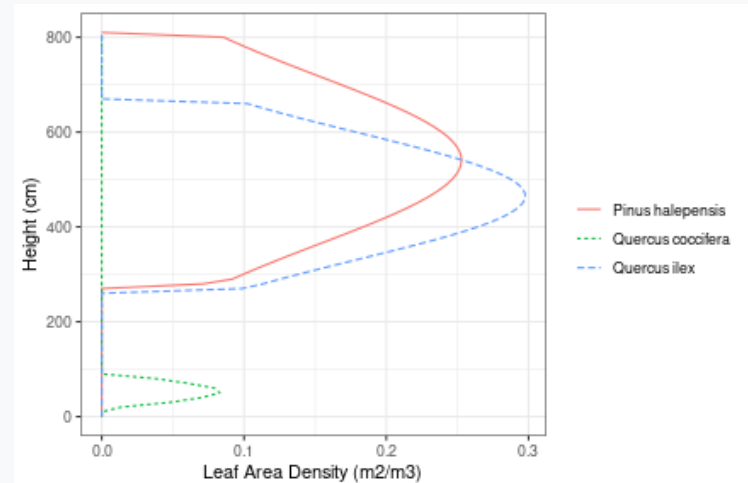
#### Leaf distribution

Vertical leaf area distribution (at the cohort-, species- or stand-level) can be examined using:

```
vprofile_leafAreaDensity(forest, SpParamsMED)
```



```
vprofile_leafAreaDensity(forest, SpParamsMED,  
  byCohorts = TRUE, bySpecies = TRUE)
```



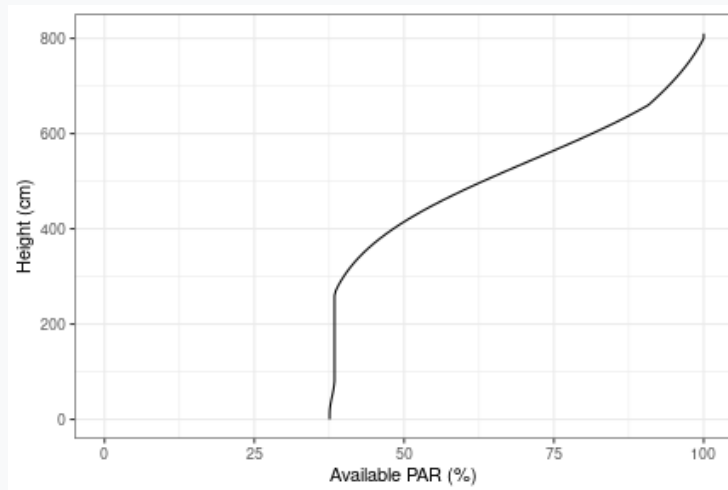


# 3. Vertical profiles

## Radiation extinction

Radiation extinction (PAR or SWR) profile across the vertical axis can also be examined:

```
vprofile_PARExtinction(forest, SpParamsMED)
```

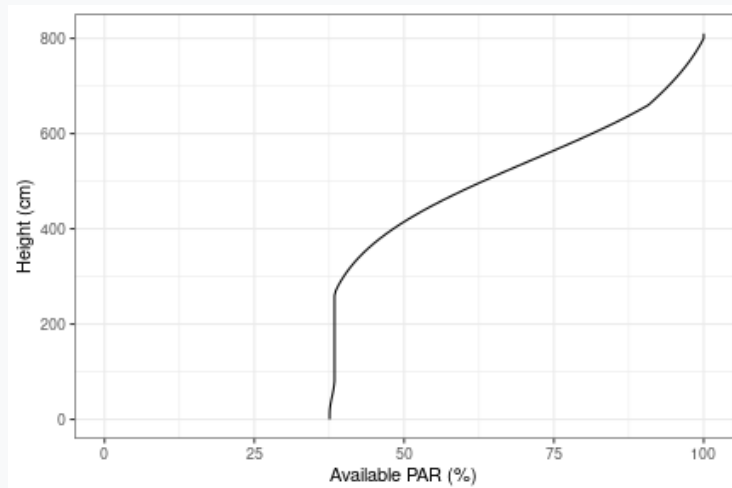


# 3. Vertical profiles

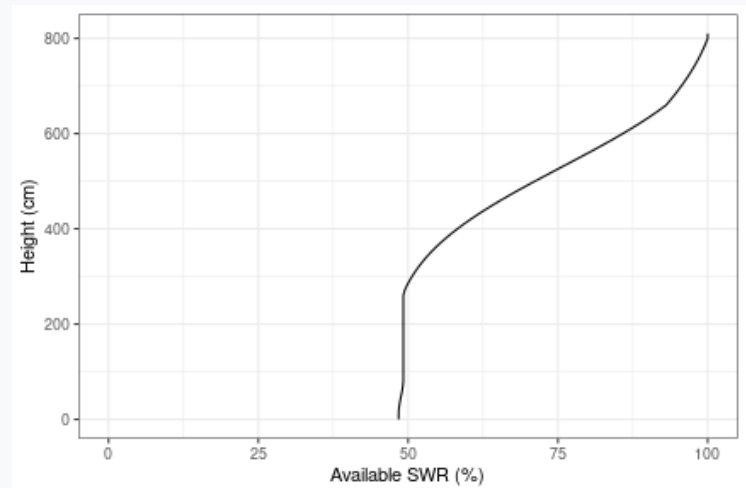
## Radiation extinction

Radiation extinction (PAR or SWR) profile across the vertical axis can also be examined:

```
vprofile_PARExtinction(forest, SpParamsMED)
```



```
vprofile_SWRExtinction(forest, SpParamsMED)
```

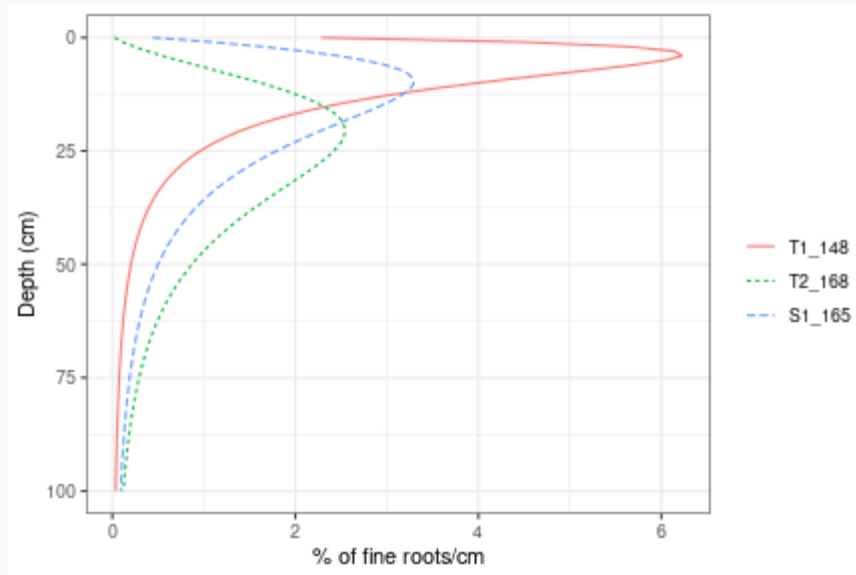


### 3. Vertical profiles

#### Belowground root distribution

Users can visually inspect the distribution of fine roots of forest objects by calling function `vprofile_rootDistribution()`:

```
vprofile_rootDistribution(forest, SpParamsMED)
```



## 3. Vertical profiles

### Interactive forest inspection

Function `shinyplot()` is a more convenient way to display properties and profiles of forest objects:

```
shinyplot(forest, SpParamsMED)
```

## 4. Soil input

### Soil physical description

Soil physical characteristics are specified using a **data.frame** with soil layers in rows and attributes in columns:

- `widths` - layer widths, in mm.
- `clay` - Percentage of clay (within volume of soil particles).
- `sand` - Percentage of sand (within volume of soil particles).
- `om` - Percentage of organic matter per dry weight (within volume of soil particles).
- `bd` - Bulk density (g/cm<sup>3</sup>)
- `r fc` - Rock fragment content (in whole-soil volume).

## 4. Soil input

### Soil physical description

Soil physical characteristics are specified using a **data.frame** with soil layers in rows and attributes in columns:

- `widths` - layer widths, in mm.
- `clay` - Percentage of clay (within volume of soil particles).
- `sand` - Percentage of sand (within volume of soil particles).
- `om` - Percentage of organic matter per dry weight (within volume of soil particles).
- `bd` - Bulk density (g/cm<sup>3</sup>)
- `r fc` - Rock fragment content (in whole-soil volume).

They can be initialized to default values using function `defaultSoilParams()`:

```
spar <- defaultSoilParams(2)
print(spar)
```

```
##   widths clay sand om   bd rfc
## 1    300   25   25 NA  1.5  25
## 2    700   25   25 NA  1.5  45
```

... and then you should modify default values according to available soil information.

## 4. Soil input

### Drawing soil physical attributes from *SoilGrids*

*SoilGrids* is a global database of soil properties:

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

## 4. Soil input

### Drawing soil physical attributes from *SoilGrids*

*SoilGrids* is a global database of soil properties:

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

Package **medfateutils** allows retrieving Soilgrids data by connecting with the SoilGrids **REST API**



## 4. Soil input

### Drawing soil physical attributes from *SoilGrids*

*SoilGrids* is a global database of soil properties:

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

Package **medfateutils** allows retrieving Soilgrids data by connecting with the SoilGrids **REST API**

To start with, we need an object of class `SpatialPoints` (from package **sp**) containing the geographic coordinates of our target forest stand:

```
cc <- matrix(c(1.32, 42.20), nrow=1)
sp <- SpatialPoints(cc, proj4string = CRS(SRS_string = "EPSG:4326"))
```

## 4. Soil input

### Drawing soil physical attributes from *SoilGrids*

*SoilGrids* is a global database of soil properties:

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

Package **medfateutils** allows retrieving Soilgrids data by connecting with the SoilGrids **REST API**

To start with, we need an object of class `SpatialPoints` (from package **sp**) containing the geographic coordinates of our target forest stand:

```
cc <- matrix(c(1.32, 42.20), nrow=1)
sp <- SpatialPoints(cc, proj4string = CRS(SRS_string = "EPSG:4326"))
```

We then call `soilgridsParams()` along with a desired vertical width (in mm) of soil layers:

```
soilgridsParams(sp, widths = c(300, 700, 1000))
```

## 4. Soil input

### Soil input object

Soil input for simulations is an object of class `soil` (a list) that is created from physical description using a function with the same name:

```
examplesoil <- soil(spar)  
class(examplesoil)
```

```
## [1] "soil" "list"
```

## 4. Soil input

### Soil input object

Soil input for simulations is an object of class `soil` (a list) that is created from physical description using a function with the same name:

```
examplesoil <- soil(spar)
class(examplesoil)
```

```
## [1] "soil" "list"
```

A `print()` function has been defined for objects of class `soil`, that displays several soil parameters and properties.

```
examplesoil
```

```
## Soil depth (mm): 1000
##
## Layer 1 [ 0 to 300 mm ]
##   clay (%): 25 silt (%): 50 sand (%): 25 organic matter (%): NA [ Silt loam ]
##   Rock fragment content (%): 25 Macroporosity (%): 5
##   Theta WP (%): 14 Theta FC (%): 30 Theta SAT (%): 49 Theta current (%) 30
##   Vol. WP (mm): 32 Vol. FC (mm): 68 Vol. SAT (mm): 111 Vol. current (mm): 68
##   Temperature (Celsius): NA
##
## Layer 2 [ 300 to 1000 mm ]
##   clay (%): 25 silt (%): 50 sand (%): 25 organic matter (%): NA [ Silt loam ]
##   Rock fragment content (%): 45 Macroporosity (%): 5
##   Theta WP (%): 14 Theta FC (%): 30 Theta SAT (%): 49 Theta current (%) 30
##   Vol. WP (mm): 55 Vol. FC (mm): 117 Vol. SAT (mm): 190 Vol. current (mm): 117
##   Temperature (Celsius): NA
##
```

## 4. Soil input

### Water retention curves

The **water retention curve** is used to represent the relationship between soil water content (  $\theta$  ; %) and soil water potential (  $\Psi$  ; MPa).

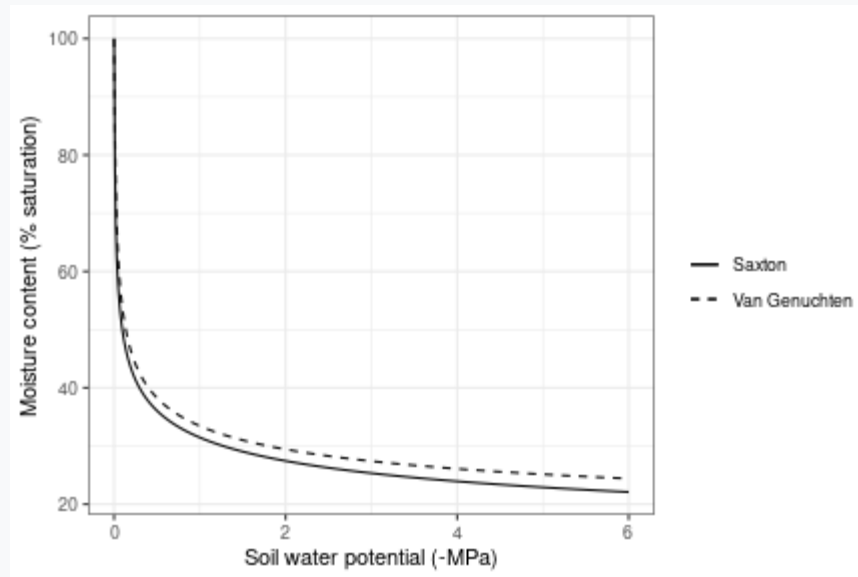
## 4. Soil input

### Water retention curves

The **water retention curve** is used to represent the relationship between soil water content ( $\theta$ ; %) and soil water potential ( $\Psi$ ; MPa).

The following code calls function `soil_retentionCurvePlot()` to illustrate the difference between the two water retention models in this soil:

```
soil_retentionCurvePlot(examplesoil, model="both")
```



## 5. Weather input

All simulations in the package require **daily weather** inputs in form of a `data.frame` with dates as `row.names`.

## 5. Weather input

All simulations in the package require **daily weather** inputs in form of a `data.frame` with dates as `row.names`.

Variables	Units
Mean/maximum/minimum temperature	$^{\circ}C$
Precipitation and potential evapo-transpiration (PET)	$l \cdot m^{-2} \cdot day^{-1}$
Mean/maximum/minimum relative humidity	%
Radiation	$MJ \cdot m^{-2} \cdot day^{-1}$
Wind speed	$m \cdot s^{-1}$



## 5. Weather input

All simulations in the package require **daily weather** inputs in form of a `data.frame` with dates as `row.names`.

Variables	Units
Mean/maximum/minimum temperature	$^{\circ}C$
Precipitation and potential evapo-transpiration (PET)	$l \cdot m^{-2} \cdot day^{-1}$
Mean/maximum/minimum relative humidity	%
Radiation	$MJ \cdot m^{-2} \cdot day^{-1}$
Wind speed	$m \cdot s^{-1}$

An example of daily weather data frame:

```
data(examplemeteo)
head(examplemeteo, 2)
```

```
##           MeanTemperature MinTemperature MaxTemperature Precipitation MeanRelativeHumidity
## 2001-01-01          3.57669         -0.5934215          6.287950          4.869109          78.73709
## 2001-01-02          1.83696         -2.3662458          4.569737          2.498292          69.70800
##           MinRelativeHumidity MaxRelativeHumidity Radiation WindSpeed WindDirection      PET
## 2001-01-01          65.15411          100.0000          12.89251          2.000000          172 1.321277
## 2001-01-02          57.43761           94.7178          13.03079          7.662544          278 2.218598
```

## 5. Weather input

All simulations in the package require **daily weather** inputs in form of a `data.frame` with dates as `row.names`.

Variables	Units
Mean/maximum/minimum temperature	$^{\circ}C$
Precipitation and potential evapo-transpiration (PET)	$l \cdot m^{-2} \cdot day^{-1}$
Mean/maximum/minimum relative humidity	%
Radiation	$MJ \cdot m^{-2} \cdot day^{-1}$
Wind speed	$m \cdot s^{-1}$

An example of daily weather data frame:

```
data(examplemeteo)
head(examplemeteo, 2)
```

```
##           MeanTemperature MinTemperature MaxTemperature Precipitation MeanRelativeHumidity
## 2001-01-01          3.57669         -0.5934215          6.287950          4.869109          78.73709
## 2001-01-02          1.83696         -2.3662458          4.569737          2.498292          69.70800
##           MinRelativeHumidity MaxRelativeHumidity Radiation WindSpeed WindDirection      PET
## 2001-01-01          65.15411          100.0000          12.89251          2.000000          172 1.321277
## 2001-01-02          57.43761          94.7178          13.03079          7.662544          278 2.218598
```

Simulation functions have been designed to accept data frames generated using package **meteoland**.

## 6. Simulation control

The behaviour of simulation models can be controlled using a set of **global parameters**.

## 6. Simulation control

The behaviour of simulation models can be controlled using a set of **global parameters**.

The default parameterization is obtained using function `defaultControl()`:

```
control <- defaultControl()
```

## 6. Simulation control

The behaviour of simulation models can be controlled using a set of **global parameters**.

The default parameterization is obtained using function `defaultControl()`:

```
control <- defaultControl()
```

A large number of control parameters exist:

```
names(control)
```

Control parameters should be left to their **default values** until their effect on simulations is fully understood!

## 7. Simulation input object

Simulation functions `spwb()` and `growth()` require combining forest, soil, species-parameter and simulation control inputs into a *single input object*.

## 7. Simulation input object

Simulation functions `spwb()` and `growth()` require combining forest, soil, species-parameter and simulation control inputs into a *single input object*.

The combination can be done via functions `forest2spwbInput()` and `forest2growthInput()`:

```
x <- forest2spwbInput(forest, examplesoil, SpParamsMED, control)
```

## 7. Simulation input object

Simulation functions `spwb()` and `growth()` require combining forest, soil, species-parameter and simulation control inputs into a *single input object*.

The combination can be done via functions `forest2spwbInput()` and `forest2growthInput()`:

```
x <- forest2spwbInput(forest, examplesoil, SpParamsMED, control)
```

Having this additional step allows *modifying the value* of specific parameters or state variables before calling the simulation functions.



## 7. Simulation input object

Simulation functions `spwb()` and `growth()` require combining forest, soil, species-parameter and simulation control inputs into a *single input object*.

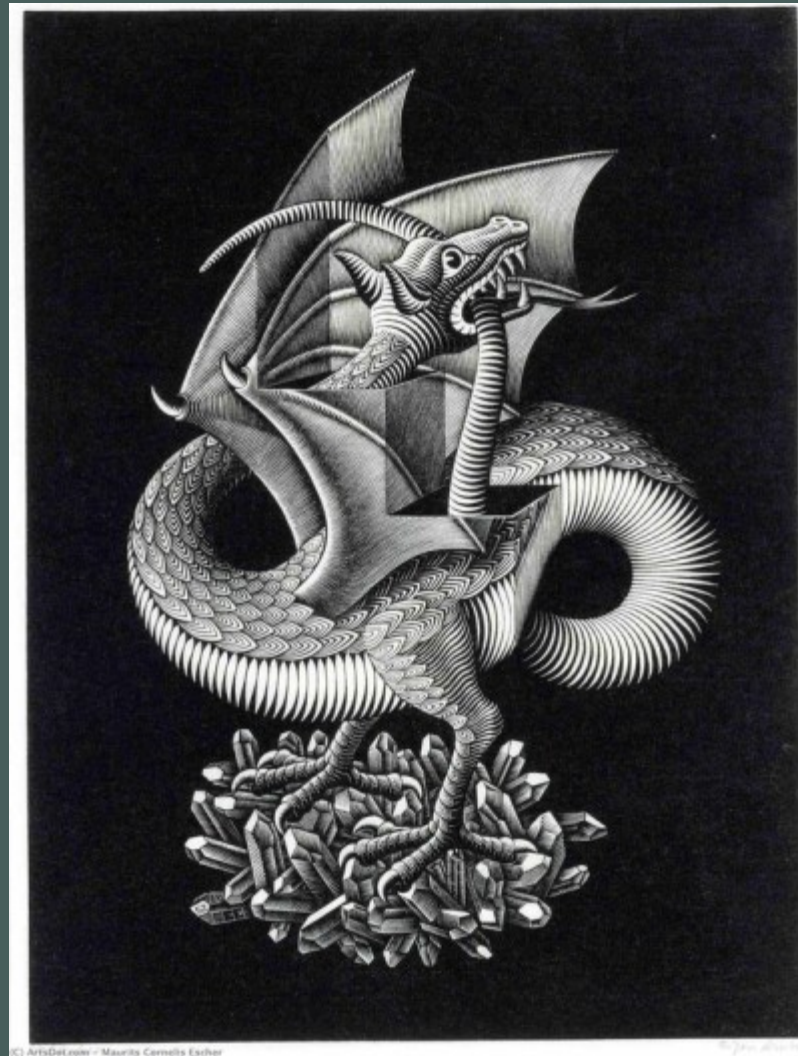
The combination can be done via functions `forest2spwbInput()` and `forest2growthInput()`:

```
x <- forest2spwbInput(forest, examplesoil, SpParamsMED, control)
```

Having this additional step allows *modifying the value* of specific parameters or state variables before calling the simulation functions.

Function `fordyn()` is different from the other two models: the user enters forest, soil, weather, species parameters and simulation control inputs *directly* into the simulation function.

## M.C. Escher - Dragon, 1952



© ArtsDot.com - Maurits Cornelis Escher