

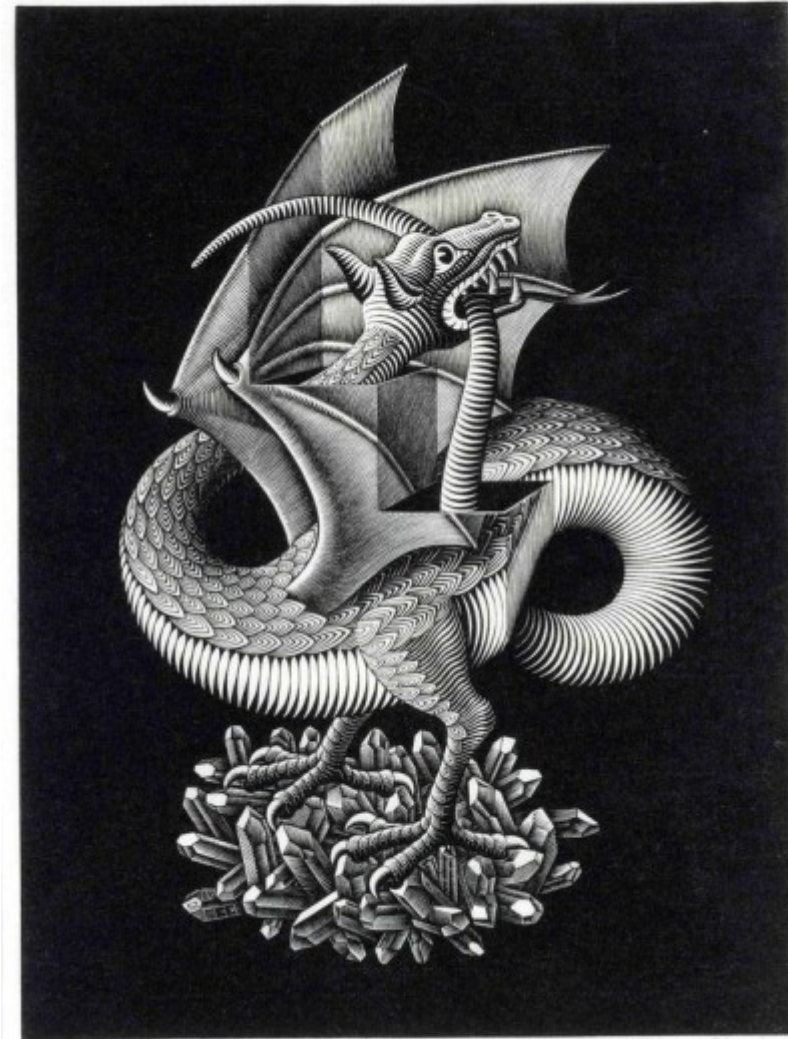
Model inputs

Miquel De Cáceres, Rodrigo Balaguer
Ecosystem Modelling Facility, CREAM

Outline

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

M.C. Escher - Dragon, 1952



(C) ArtisDiel.com - Maarten Cornelis Escher

1. Species parameters

Species parameter table

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.

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

A large number of parameters (157 columns) can be found in `SpParamsMED`, which may be intimidating.

You can find parameter definitions in table `SpParamsDefinition`:

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

Species parameter table

The following table shows parameter definitions and units:

Show

4

 entries

Search:

	ParameterName	ParameterGroup	Definition	Type	Units	Strict
1	Name	Identity	Plant names (species binomials, genus or other) used in vegetation data	String		true
2	SpIndex	Identity	Internal species codification (0,1,2,)	Integer		true
3	AcceptedName	Taxonomic identity	Accepted scientific name of a taxon (genus, species, subspecies or variety) used for parameterization	String		false
4	Species	Taxonomic identity	Taxonomic species of accepted name	String		false

Showing 1 to 4 of 156 entries

Previous

1

2

3

4

5

...

39

Next

2. Forest input

Forest class

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

```
1 forest <- medfate::exampleforest
```

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

```
1 forest$treeData
```

	Species	N	DBH	Height	Z50	Z95
1	Pinus halepensis	168	37.55	800	100	600
2	Quercus ilex	384	14.60	660	300	1000

and `shrubData` (for shrubs):

```
1 forest$shrubData
```

	Species	Cover	Height	Z50	Z95
1	Quercus coccifera	3.75	80	200	1000

Forest class

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*).

Creating a ‘forest’ 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 **medfate** provides functions that may be helpful:

Function	Description
<code>forest_mapShrubTable()</code>	Helps filling <code>shrubData</code> table
<code>forest_mapTreeTable()</code>	Helps filling <code>treeData</code> table
<code>forest_mapWoodyTables()</code>	Helps filling a <code>forest</code> object

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).

<pre>1 plant_basalArea(forest, SpParamsMED)</pre>	<pre>1 stand_basalArea(forest)</pre>
<pre>T1_148 T2_168 S1_165 18.604547 6.428755 NA</pre>	<pre>[1] 25.0333</pre>
<pre>1 plant_LAI(forest, SpParamsMED)</pre>	<pre>1 stand_LAI(forest, SpParamsMED)</pre>
<pre>T1_148 T2_168 S1_165 0.84874773 0.70557382 0.03062604</pre>	<pre>[1] 1.758585</pre>

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()`:

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

	SP	N	DBH	Cover	H	CR	LAI_live	LAI_expanded	LAI_dead
T1_148	148	168.0000	37.55	NA	800	0.6605196	0.84874773	0.84874773	0
T2_168	168	384.0000	14.60	NA	660	0.6055642	0.70557382	0.70557382	0
S1_165	165	749.4923	NA	3.75	80	0.8032817	0.03062604	0.03062604	0
	LAI_nocomp	ObsID							
T1_148	1.29720268	<NA>							
T2_168	1.01943205	<NA>							
S1_165	0.04412896	<NA>							

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

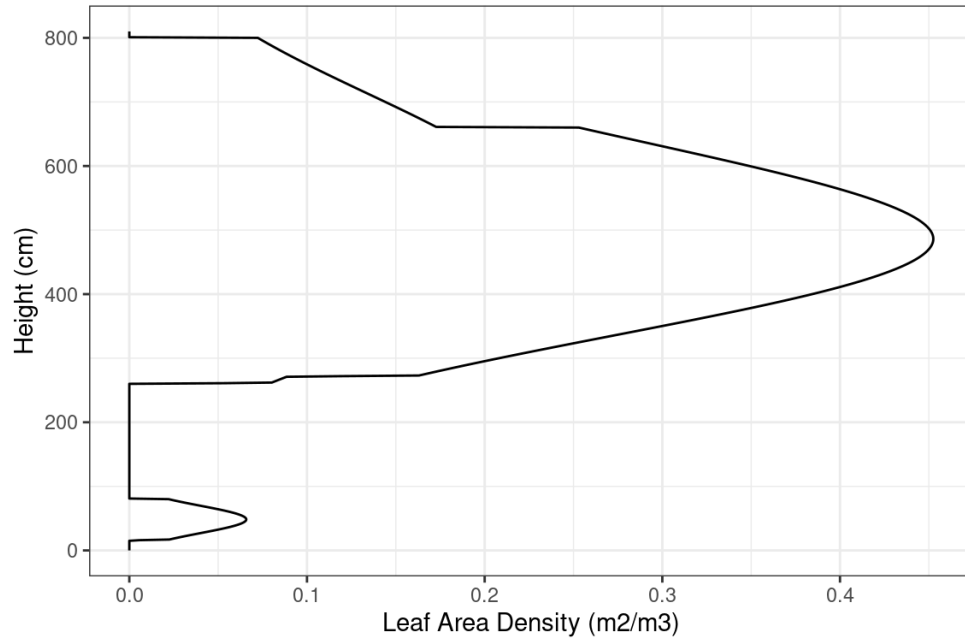
Users will not normally call `forest2aboveground()`, but is important to understand what is going on behind the scenes.

3. Vertical profiles

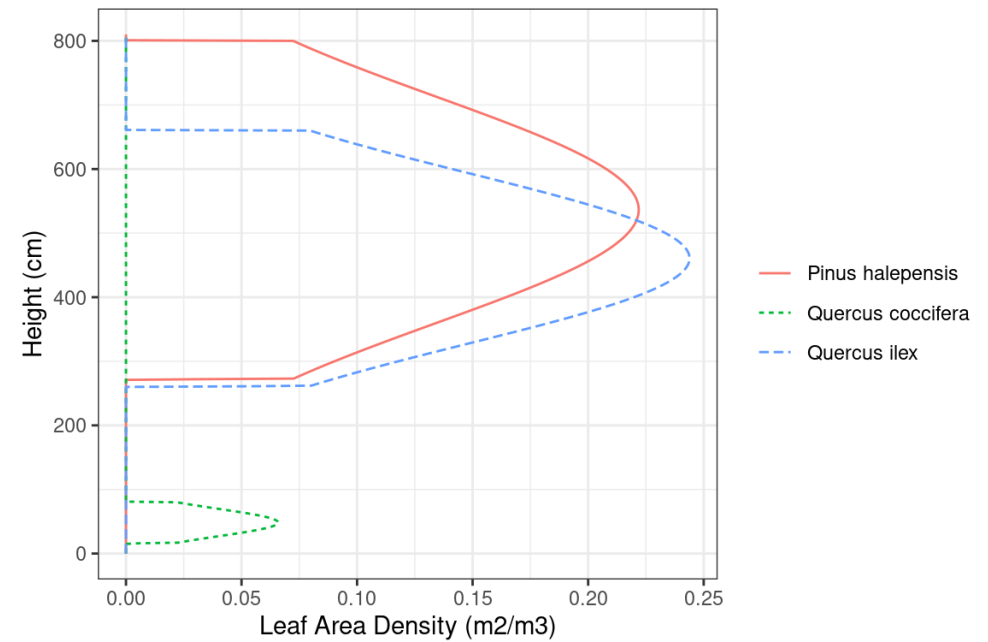
Leaf distribution

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

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



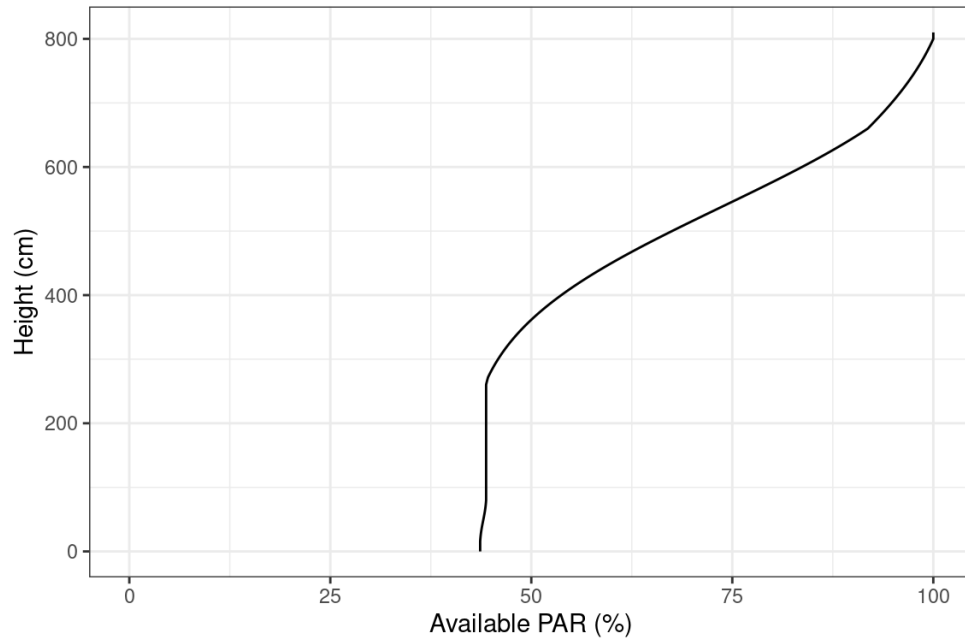
```
1 vprofile_leafAreaDensity(forest, SpParamsMED,  
2   byCohorts = TRUE, bySpecies = TRUE)
```



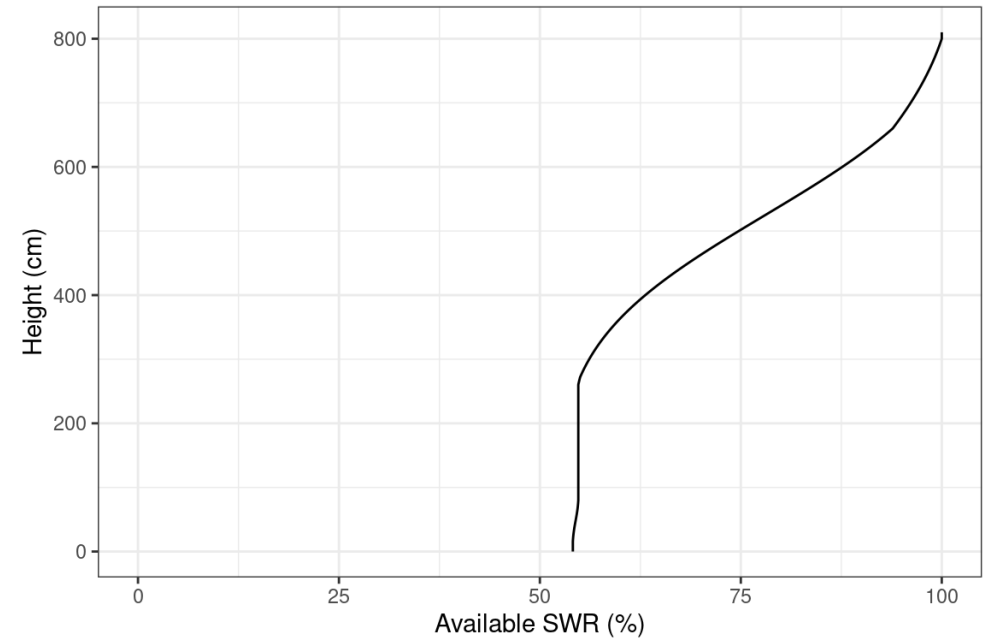
Radiation extinction

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

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



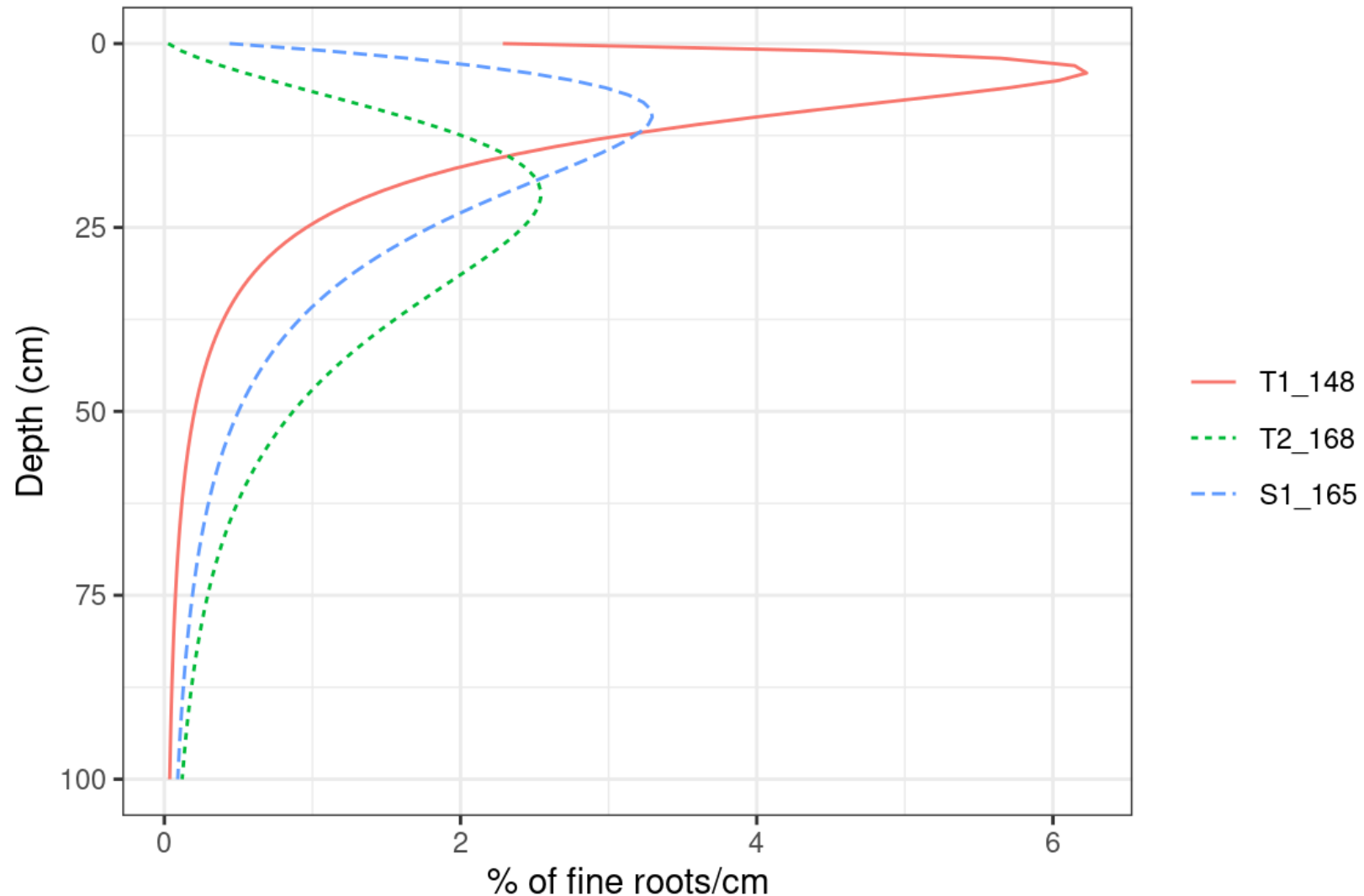
```
1 vprofile_SWRExtinction(forest, SpParamsMED)
```



Belowground root distribution

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

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



Interactive forest inspection

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

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


4. Soil input

Soil physical description

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

Attribute	Description
<code>widths</code>	Layer widths, in mm.
<code>clay</code>	Percentage of clay (within volume of soil particles).
<code>sand</code>	Percentage of sand (within volume of soil particles).
<code>om</code>	Percentage of organic matter per dry weight (within volume of soil particles).
<code>nitrogen</code>	Total nitrogen (g/kg). Not used at present.
<code>bd</code>	Bulk density (g/cm3)
<code>rfc</code>	Rock fragment content (in whole-soil volume).

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

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



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

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

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 **medfateland** allows retrieving Soilgrids data by connecting with the SoilGrids [REST API](#)

To start with, we need a spatial object of class `sf` or `sfc` (from package `sf`) containing the geographic coordinates of our target forest stand:

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

Initialized soil

The soil initialized for simulations is a data frame of class `soil` that is created from physical description using a function with the same name:

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

```
[1] "soil"      "data.frame"
```

The initialised soil data frame contains additional columns with soil hydraulic parameters and state variables for moisture (`w`) and temperature (`Temp`):

```
1 examplesoil
```

	widths	sand	clay	usda	om	nitrogen	bd	rhc	macro	Ksat	VG_alpha	
1	300	25	25	Silt	loam	NA	NA	1.5	25	0.0485	5401.471	89.16112
2	700	25	25	Silt	loam	NA	NA	1.5	45	0.0485	5401.471	89.16112
	VG_n	VG_theta_res	VG_theta_sat	W	Temp							
1	1.303861	0.041	0.423715	1	NA							
2	1.303861	0.041	0.423715	1	NA							

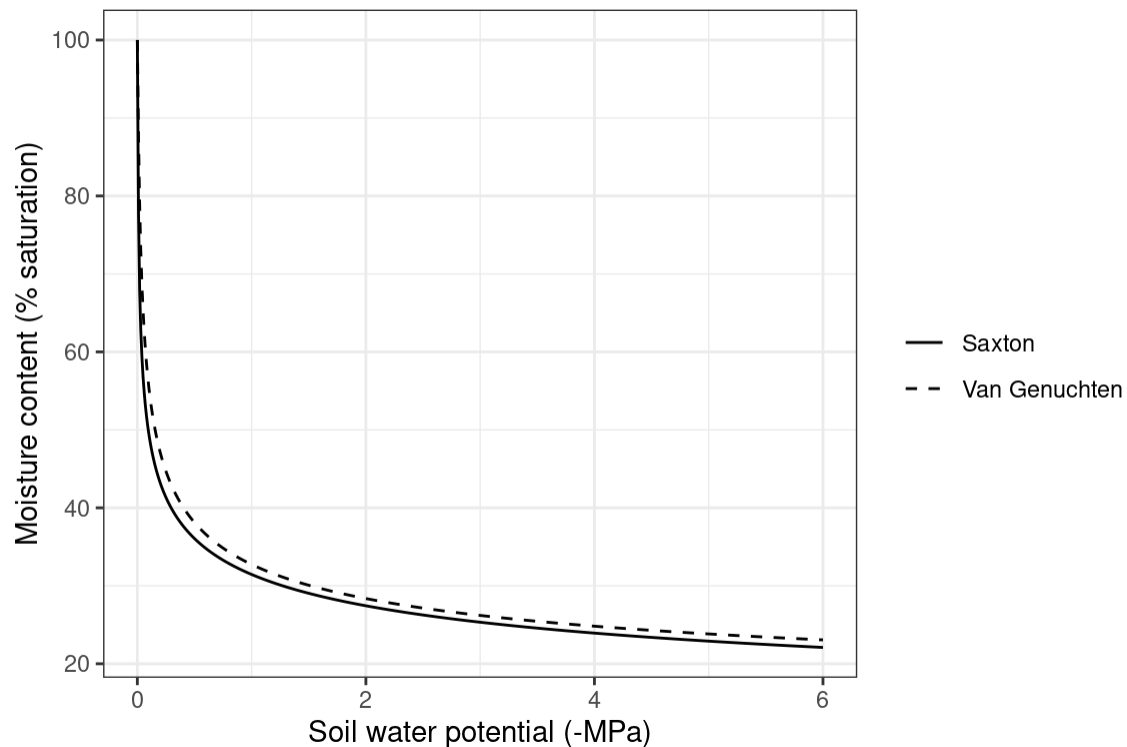
We can skip calling function `soil()` in our scripts to run simulations, but again is good to know what is behind the scenes.

Water retention curves

The **water retention curve** is used to represent the relationship between soil water content (θ ; %) and soil water potential (Ψ ; MPa).

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

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



5. Simulation control

Simulation control list

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

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

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

A large number of control parameters exist:

```
1 names(control)
```

Important

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

6. Simulation input object

Simulation input object

Functions `spwb()` and `growth()`

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 `spwbInput()` and `growthInput()`:

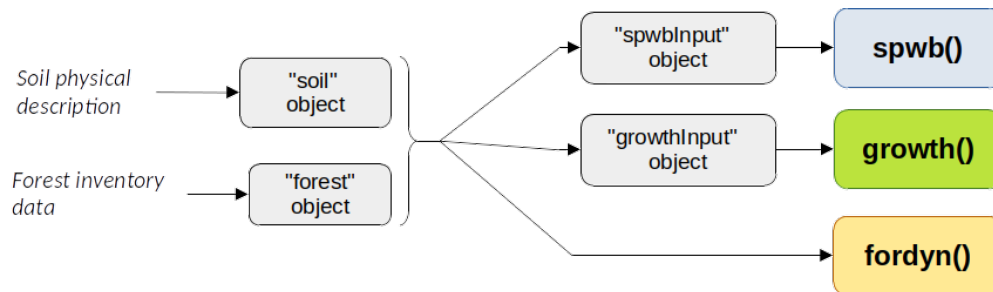
```
1 x <- spwbInput(forest, examplesoil, SpParamsMED, control)
```

Function `fordyn()`

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

Summary

The following workflow summarises the initialisation for the three functions:



7. Weather forcing

Weather data frame

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

Variables	Units
Maximum/minimum temperature	$^{\circ}C$
Precipitation	$l \cdot m^{-2} \cdot day^{-1}$
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 is included in package **medfate**:

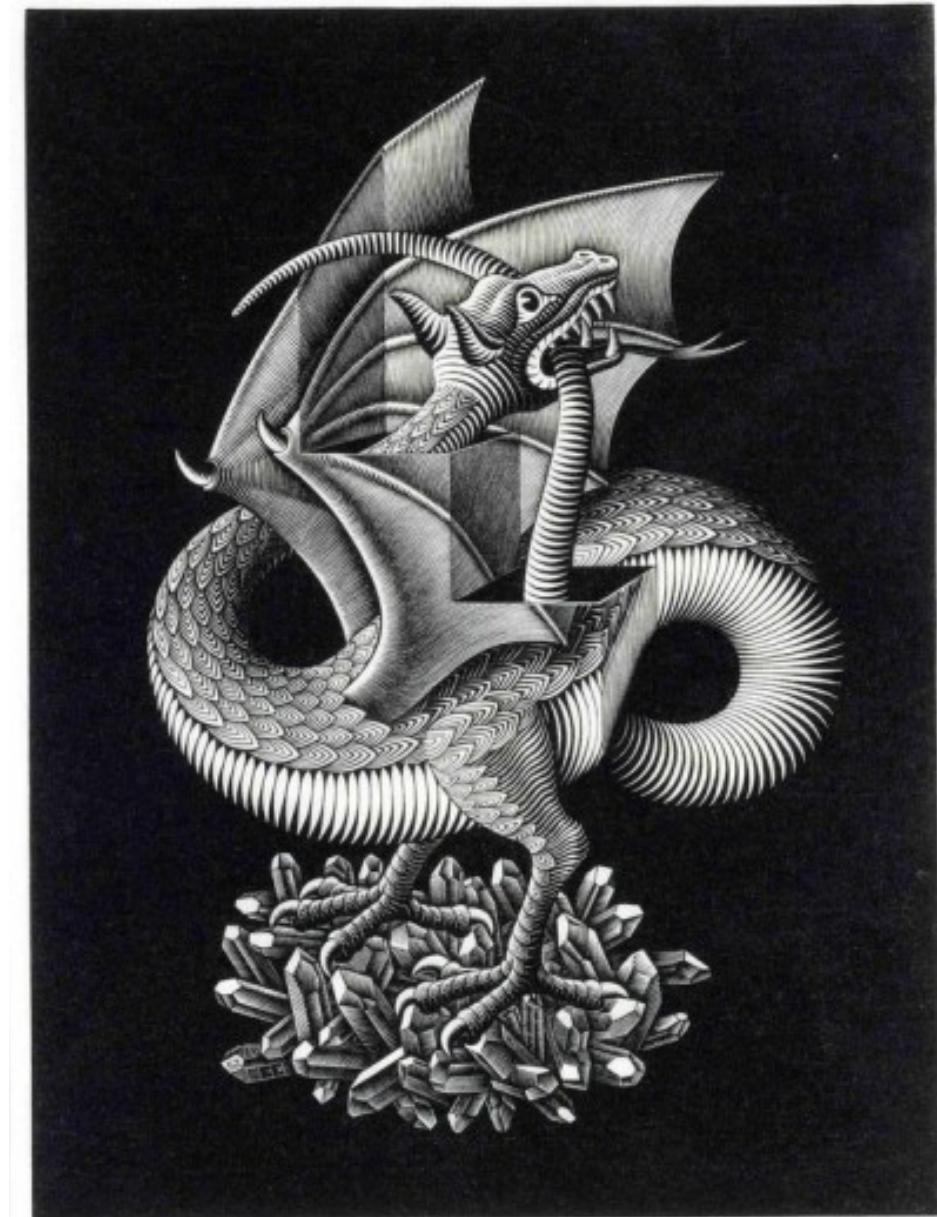
```
1 data(examplemeteo)
2 head(examplemeteo, 2)
```

	dates	MinTemperature	MaxTemperature	Precipitation	MinRelativeHumidity
1	2001-01-01	-0.5934215	6.287950	4.869109	65.15411
2	2001-01-02	-2.3662458	4.569737	2.498292	57.43761
		MaxRelativeHumidity	Radiation	WindSpeed	
1		100.0000	12.89251	2.000000	
2		94.7178	13.03079	7.662544	

Tip

- Simulation functions have been designed to accept data frames generated using package [meteoland](#).
- The package will try to fill missing values of some variables with estimates (e.g. for radiation or relative humidity).
- Additional variables (atmospheric pressure or CO2 concentration) are optional.

M.C. Escher - Dragon, 1952



(C) ArtsDot.com - M.C. Escher

The Print Collector