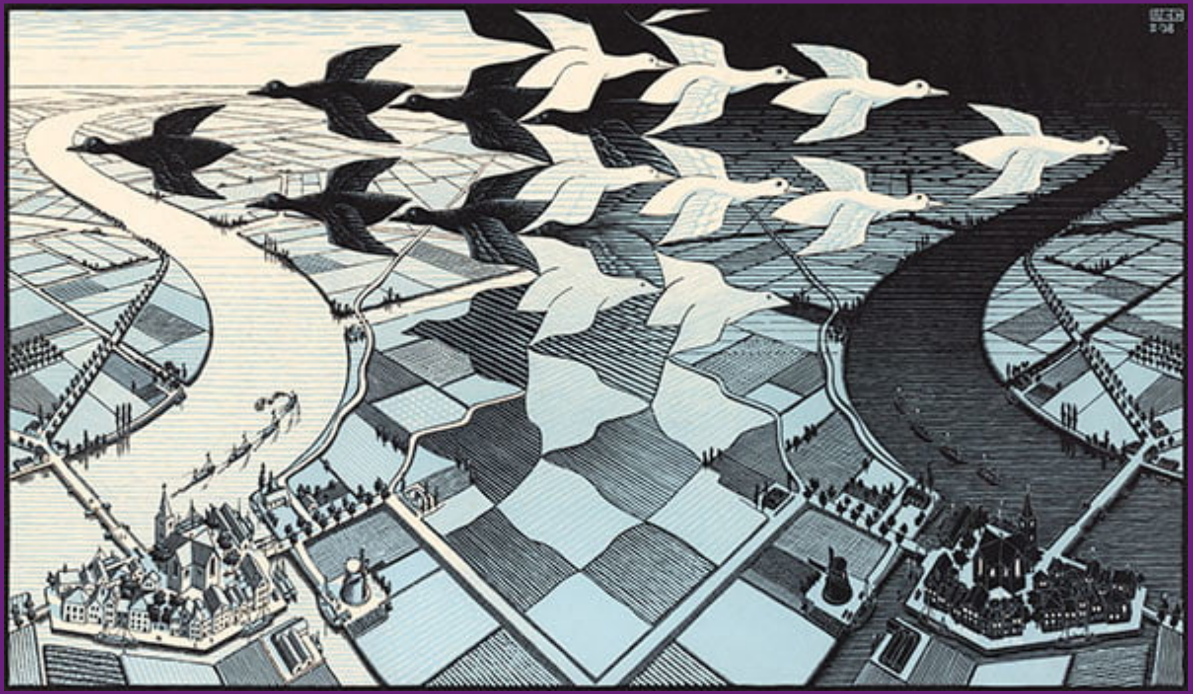
Forest water and energy balance (practice)

Miquel De Cáceres, Rodrigo Balaguer

Ecosystem Modelling Facility, CREAF

Outline

1. Water balance input object
2. Basic water balance
3. Evaluating model performance
4. Advanced water/energy balance
5. Modifying model inputs



M.C. Escher - Night and day, 1938

1. Water balance input object

Creating the water balance input object

We assume we have an appropriate `forest` object and species parameter data frame:

```r
data(exampleforest)
data(SpParamsMED)
```

a soil description data frame:

```r
examplesoil <- defaultSoilParams(4)
```

and a simulation control list:

```r
control <- defaultControl(transpirationMode = "Granier", soilDomains = "buckets")
```

Important

- Plant transpiration (`transpirationMode`) should be `"Granier"` (basic), `"Sperry"` (advanced with Sperry) or `"Sureau"` (advanced with Sureau-ECOS).
- Soil water movement (`soilDomains`) should be `"buckets"` (multi-bucket), `"single"` (single-domain) or `"dual"` (dual-permeability).

With these four elements we can build our input object for function `spwb()`:

```r
x <- spwbInput(exampleforest, examplesoil, SpParamsMED, control)
```

Structure of the water balance input object (1)

The water balance input object is a `list` with several elements:

```r
names(x)
```

```
 [1] "control"                 "soil"
 [3] "snowpack"                "canopy"
 [5] "herbLAI"                 "herbLAImax"
 [7] "cohorts"                 "above"
 [9] "below"                   "belowLayers"
[11] "paramsPhenology"         "paramsAnatomy"
[13] "paramsInterception"      "paramsTranspiration"
[15] "paramsWaterStorage"      "internalPhenology"
[17] "internalWater"           "internalLAIDistribution"
[19] "internalFCCS"
```

Element `soil` contains the (initialized) soil data frame:

```r
x$soil
```

```
  widths sand clay      usda om nitrogen  bd rfc  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
3   1000   25   25 Silt loam NA       NA 1.5  75 0.0485 5401.471 89.16112
4   2000   25   25 Silt loam NA       NA 1.5  95 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
3 1.303861        0.041     0.423715 1   NA
4 1.303861        0.041     0.423715 1   NA
```

Element `cohorts` contains the species identity of each cohort:

```r
x$cohorts
```

```
        SP              Name
T1_148 148  Pinus halepensis
T2_168 168      Quercus ilex
S1_165 165 Quercus coccifera
```

Structure of the water balance input object (2)

Element `above` contains above-ground description of vegetation:

```r
x$above
```

```
          H        CR   LAI_live LAI_expanded LAI_dead ObsID
T1_148 800 0.6605196 0.84874773   0.84874773        0  <NA>
T2_168 660 0.6055642 0.70557382   0.70557382        0  <NA>
S1_165  80 0.8032817 0.03062604   0.03062604        0  <NA>
```

Element `below` contains below-ground description of vegetation:

```r
x$below
```

```
       Z50  Z95 Z100
T1_148 100  600   NA
T2_168 300 1000   NA
S1_165 200 1000   NA
```

Elements `params*` contain cohort-level parameters, for example…

```r
x$paramsTranspiration
```

```
            Gswmin  Tmax_LAI   Tmax_LAIsq Psi_Extract Exp_Extract  VCleaf_c
T1_148 0.003086667 0.1869849 -0.008372458  -0.9218219    1.504542 11.137050
T2_168 0.004473333 0.1251027 -0.005601615  -1.9726871    1.149052  1.339370
S1_165 0.010455247 0.1340000 -0.006000000  -2.1210726    1.300000  2.254991
         VCleaf_d  VCstem_c  VCstem_d      WUE   WUE_par     WUE_co2    WUE_vpd
T1_148 -2.380849 12.709999 -5.290000 8.525550 0.5239136 0.002586327 -0.2647169
T2_168 -2.582279  3.560000 -7.720000 8.968208 0.1412266 0.002413091 -0.5664879
S1_165 -3.133381  3.095442 -7.857378 7.900000 0.3643000 0.002757000 -0.4636000
```

2. Basic water balance

Water balance run

Let us assume we have an appropriate weather data frame:

```r
data(examplemeteo)
```

The call to function `spwb()` needs the water balance input object, the weather data frame, latitude and elevation:

```r
S <- spwb(x, examplemeteo, latitude = 41.82592, elevation = 100)
```

```
Initial plant water content (mm): 4.73001
Initial soil water content (mm): 290.875
Initial snowpack content (mm): 0
Performing daily simulations

 [Year 2001]:............

Final plant water content (mm): 4.7285
Final soil water content (mm): 274.723
Final snowpack content (mm): 0
Change in plant water content (mm): -0.00151775
Plant water balance result (mm): -0.00151775
Change in soil water content (mm): -16.1521
Soil water balance result (mm): -16.1521
Change in snowpack water content (mm): 0
Snowpack water balance result (mm): -7.10543e-15
Water balance components:
  Precipitation (mm) 513 Rain (mm) 462 Snow (mm) 51
  Interception (mm) 92 Net rainfall (mm) 370
  Infiltration (mm) 400 Infiltration excess (mm) 21 Saturation excess (mm) 0 Capillarity rise (mm) 0
  Soil evaporation (mm) 26  Herbaceous transpiration (mm) 14 Woody plant transpiration (mm) 249
  Plant extraction from soil (mm) 249  Plant water balance (mm) -0 Hydraulic redistribution (mm) 5
  Runoff (mm) 21 Deep drainage (mm) 128
```

Water balance output object (1)

Function `spwb()` returns an object of class with the same name, actually a list:

```r
class(S)
```

```
[1] "spwb" "list"
```

It is interesting to inspect the list element names:

```r
names(S)
```

```
 [1] "latitude"     "topography"   "weather"      "spwbInput"    "spwbOutput"  
 [6] "WaterBalance" "Soil"         "Snow"         "Stand"        "Plants"      
```

Elements	Information
`latitude`, `topography`, `weather`, `spwbInput`	Copies of the information used in the call to `spwb()`
`spwbOutput`	State variables at the end of the simulation (can be used as input to a subsequent one)
`WaterBalance`, `Soil`, `Snow`, `Stand`, `Plants`	Daily outputs

Water balance output object (2)

Daily outputs are `data.frame` objects with **dates as row names** and **variables in columns**, for example:

```r
head(S$WaterBalance, 2)
```

```
                 PET Precipitation     Rain Snow  NetRain Snowmelt Infiltration
2001-01-01 0.8828475      4.869109 4.869109    0 3.424180        0     3.424180
2001-01-02 1.6375337      2.498292 2.498292    0 1.071747        0     1.071747
           InfiltrationExcess SaturationExcess Runoff DeepDrainage
2001-01-01                  0                0      0    2.7617207
2001-01-02                  0                0      0    0.1895324
           CapillarityRise Evapotranspiration Interception SoilEvaporation
2001-01-01               0           2.107388     1.444929       0.4478948
2001-01-02               0           2.324525     1.426545       0.5000000
           HerbTranspiration PlantExtraction Transpiration
2001-01-01        0.01102343       0.2035406     0.2035406
2001-01-02        0.02044661       0.3775336     0.3775336
           HydraulicRedistribution
2001-01-01                       0
2001-01-02                       0
```

`Soil` is itself a list with several data frames with different results by soil layer:

```r
names(S$Soil)
```

```
[1] "SWC"            "RWC"            "REW"            "ML"
[5] "Psi"            "PlantExt"       "HydraulicInput"
```

Likewise, `Plants` is itself a list with several data frames with different results by cohort:

```r
names(S$Plants)
```

```
 [1] "LAI"                 "LAIlive"             "FPAR"
 [4] "AbsorbedSWRFraction" "Transpiration"       "GrossPhotosynthesis"
 [7] "PlantPsi"            "LeafPLC"             "StemPLC"
[10] "PlantWaterBalance"   "LeafRWC"             "StemRWC"
[13] "LFMC"                "PlantStress"
```

Accessing and summarizing model result

Summary function

The package provides a `summary()` function for objects of class `spwb`. It can be used to extract/summarize the model's output at different temporal steps (i.e. weekly, annual, …).

For example, to aggregate water balance results by months one can use:

```r
summary(S, freq="months",FUN=sum, output="WaterBalance")
```

Parameter `output` indicates the element of the `spwb` object for which we desire a summary. Similarly, it is possible to calculate the average stress of plant cohorts by months:

```r
summary(S, freq="months",FUN=mean, output="PlantStress")
```

Extraction function

Post-processing is much more convenient using function `extract()` which extracts model results in a format compatible with **tidyverse** manipulation.

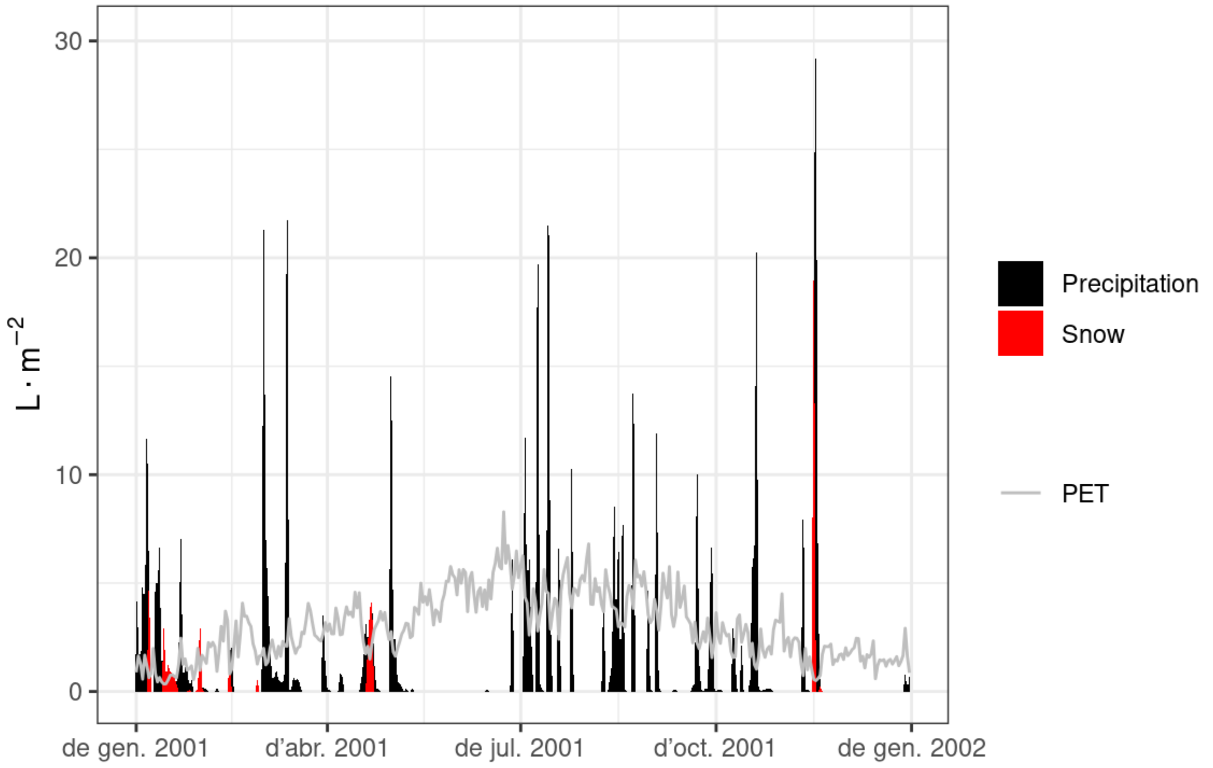
For example, the following returns all stand level results by date:

```r
extract(S, level = "forest")
```

Plotting

The package provides a `plot()` function for objects of class `spwb`. It can be used to show weather inputs and different components of the water balance, for example:

```r
plot(S, type = "PET_Precipitation")
```



The help page of `?plot.spwb` lists all the possible plots…

… but instead of typing all plots, we can call the interactive plot function and explore them all:

```r
shinyplot(S)
```

3. Evaluating model performance

Observed data and evaluation metrics

The package provides functions to compare predictions with observations (use `?evaluation` for details on how observations should be arranged).

The package includes a (fake) example data set of observed data:

```
       dates       SWC       ETR   E_T1_148   E_T2_168 FMC_T1_148 FMC_T2_168
1 2001-01-01 0.3007733 2.2436218 0.09187857 0.14142950   125.9071   93.07915
2 2001-01-02 0.3091627 2.3236565 0.26480973 0.19095008   125.9137   93.07863
3 2001-01-03 0.2996498 0.7409083 0.15345643 0.17546363   125.8760   93.10512
4 2001-01-04 0.3042764 1.7173522 0.23470647 0.04643454   125.8643   93.07022
5 2001-01-05 0.3054886 2.0002562 0.37687792 0.10623552   125.8493   93.08487
6 2001-01-06 0.3062005 2.0722706 0.16342360 0.05550329   125.9367   93.07343
    BAI_T1_148 BAI_T2_168    DI_T1_148 DI_T2_168
1 6.222625e-06          0 9.948881e-08         0
2 3.091274e-10          0 1.071090e-11         0
3 1.298482e-13          0 0.000000e+00         0
4 2.886195e-11          0 5.552753e-13         0
5 1.287020e-03          0 1.367289e-05         0
6 1.471202e-03          0 1.000411e-05         0
```

Note the observation dates in `dates` column. The remaining variables are observations to be matched with simulation results.

A single evaluation metric for soil water content can be calculated using:

```
evaluation_metric(S, exampleobs, type = "SWC", metric = "MAE")
```

```
[1] 0.005002766
```

or many of them:

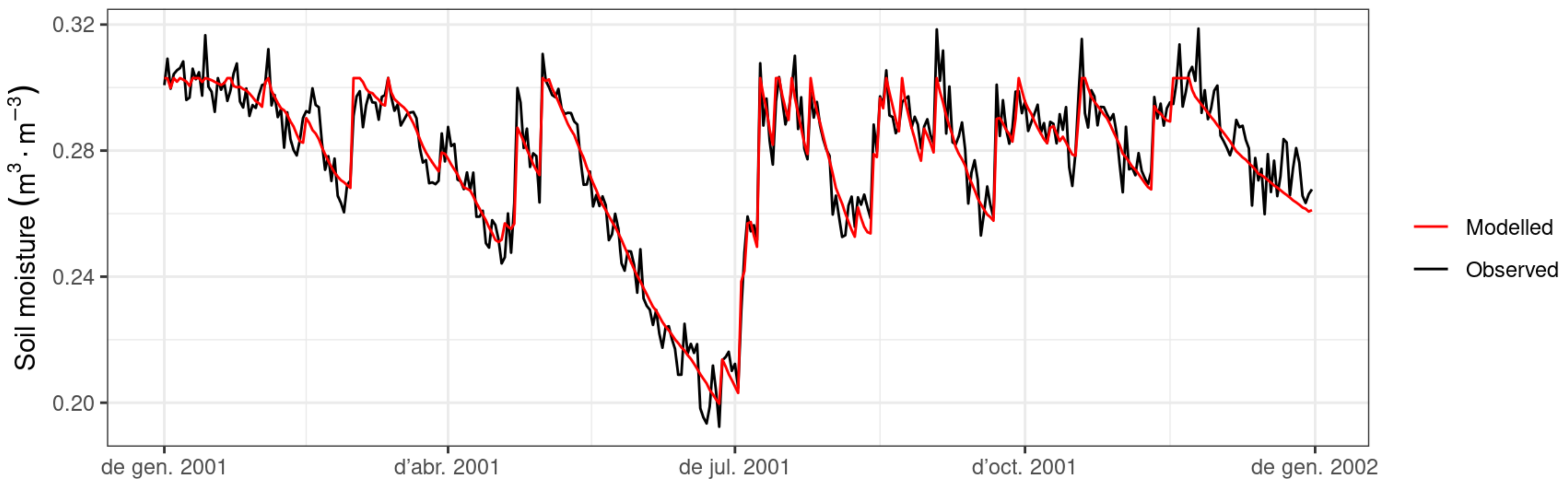
```
evaluation_stats(S, exampleobs, type = "SWC")
```

```
            n          Bias      Bias.rel           MAE       MAE.rel 
 3.650000e+02 -5.419844e-04 -1.954972e-01  5.002766e-03  1.804529e+00 
            r           NSE       NSE.abs 
 9.683900e-01  9.372955e-01  7.378057e-01 
```

Evaluation plots and interactive evaluation

Evaluation functions also allow visualizing the comparison as time series or scatter plots:

```r
evaluation_plot(S, exampleobs, type = "SWC", plotType = "dynamics")
```



Alternatively, the observed data can be supplied as an additional parameter to `shinyplot()` for interactive graphics including model evaluation:

```r
shinyplot(S, exampleobs)
```

4. Advanced water/energy balance

Creating an input object for the advanced model

The most important step to run the advanced model is to specify the appropriate transpiration mode in the `control` parameters:

```r
control <- defaultControl("Sperry")
```

If we want to plot sub-daily results, we must specify it as follows:

```r
control$subdailyResults <- TRUE
```

We can build our input object for function `spwb()` using the same function as before:

```r
x_adv <- spwbInput(exampleforest, examplesoil, SpParamsMED, control)
```

The water balance input object contains the same elements…

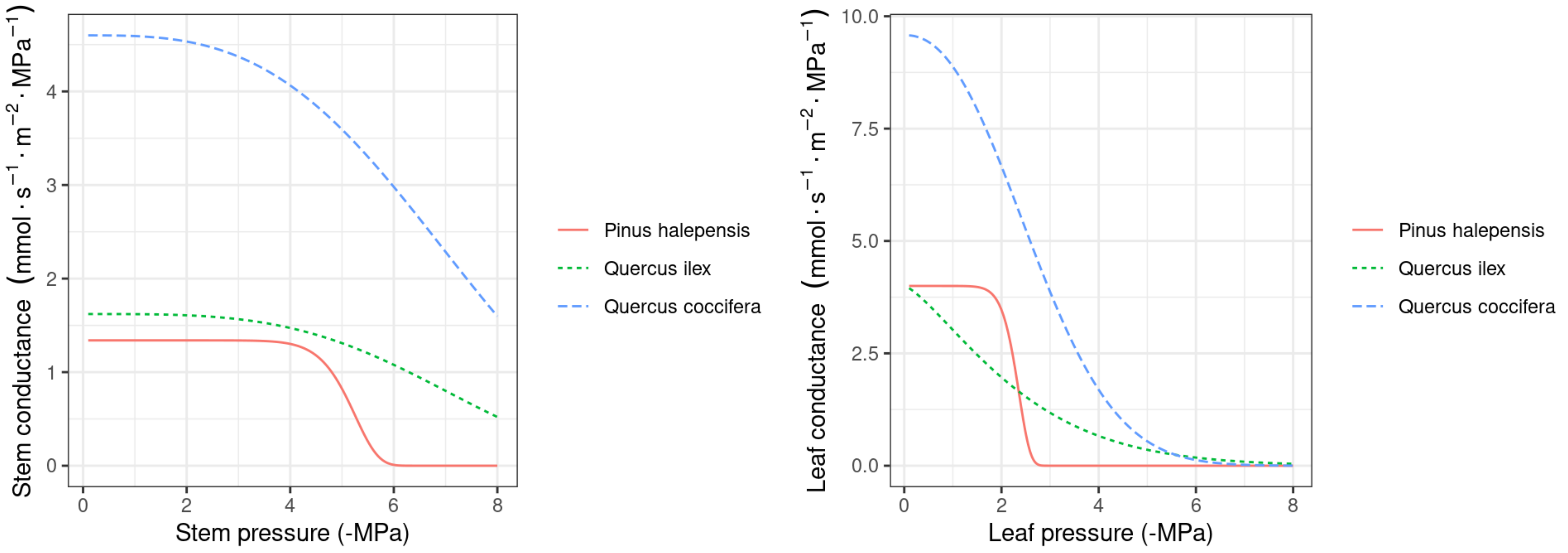
```r
names(x_adv)
```

```
 [1] "control"                "soil"                  
 [3] "snowpack"               "canopy"                
 [5] "herbLAI"                "herbLAImax"            
 [7] "cohorts"                "above"                 
 [9] "below"                  "belowLayers"           
[11] "paramsPhenology"        "paramsAnatomy"         
[13] "paramsInterception"     "paramsTranspiration"   
[15] "paramsWaterStorage"     "internalPhenology"     
[17] "internalWater"          "internalLAIDistribution"
[19] "internalFCCS"          
```

Vulnerability curves and supply functions

We can inspect *hydraulic vulnerability curves* (i.e. how hydraulic conductance of a given segment changes with the water potential) for each plant cohort and each of the different segments of the soil-plant hydraulic network:

```r
g1 <- hydraulics_vulnerabilityCurvePlot(x_adv, type="stem", speciesNames = TRUE)
g2 <- hydraulics_vulnerabilityCurvePlot(x_adv, type="leaf", speciesNames = TRUE)
cowplot::plot_grid(g1, g2, ncol = 2)
```



Water/energy balance run for a single day (1)

Since the model operates at a daily and sub-daily temporal scales, it is possible to perform soil water balance for one day only. First we need a weather vector:

```r
d = 100
meteovec <- unlist(examplemeteo[d,-1])
meteovec
```

```
      MinTemperature       MaxTemperature        Precipitation MinRelativeHumidity
           0.3881289           10.0320962            0.0000000          42.0207334
 MaxRelativeHumidity            Radiation            WindSpeed
          82.3036989           28.7201692            3.3228840
```

and a string with the target date:

```
[1] "2001-04-10"
```

At this point, we can call function `spwb_day()`

```r
sd1<-spwb_day(x_adv, date, meteovec,
             latitude = 41.82592, elevation = 100,
             slope= 0, aspect = 0)
```

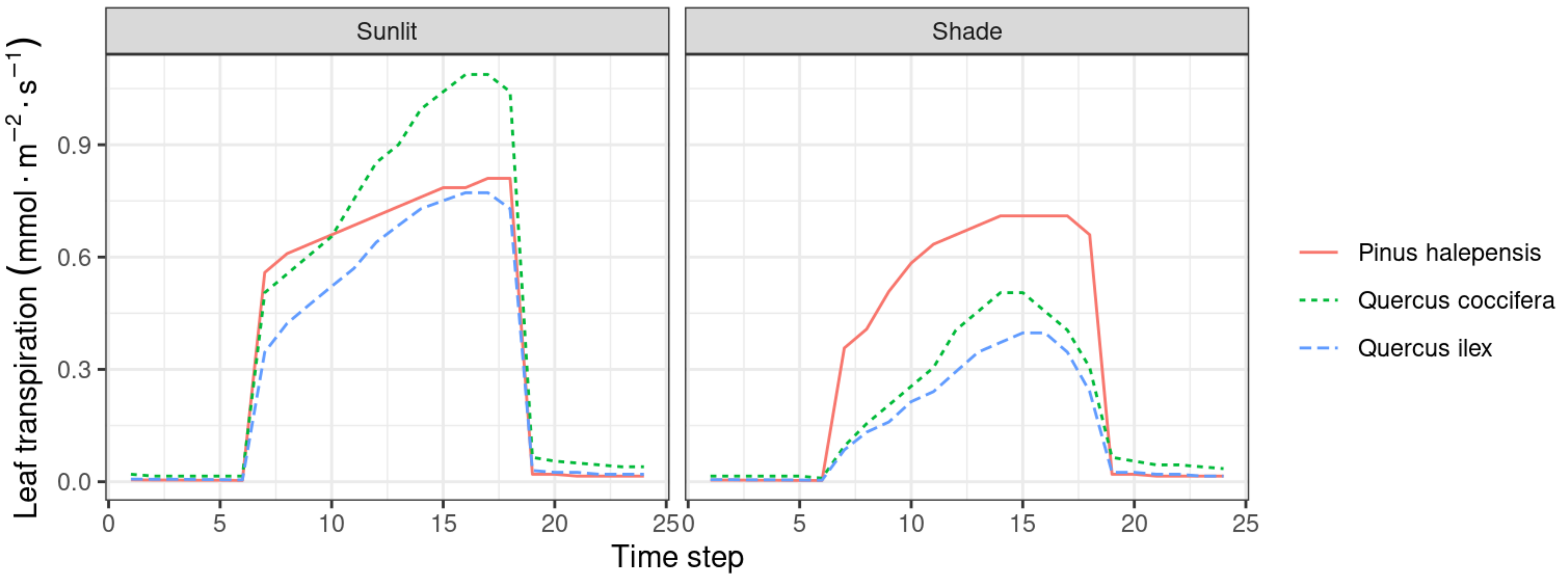
Warning

By default, a call to `spwb_day()` will modify the input object. This behavior can be deactivated by using `modifyInput = FALSE` in the simulation call (see also `?resetInputs`).

Water/energy balance run for a single day (2)

As with `spwb()`, there is a plot function for results of `spwb_day()`. For example we can use:

```r
plot(sd1, type = "LeafTranspiration", bySpecies = TRUE)
```



More conveniently, you can examine multiple plots interactively:

```r
shinyplot(sd1)
```

Water/energy balance run for multiple days

We can now run the advanced water balance model (which takes 15 sec. aprox.)

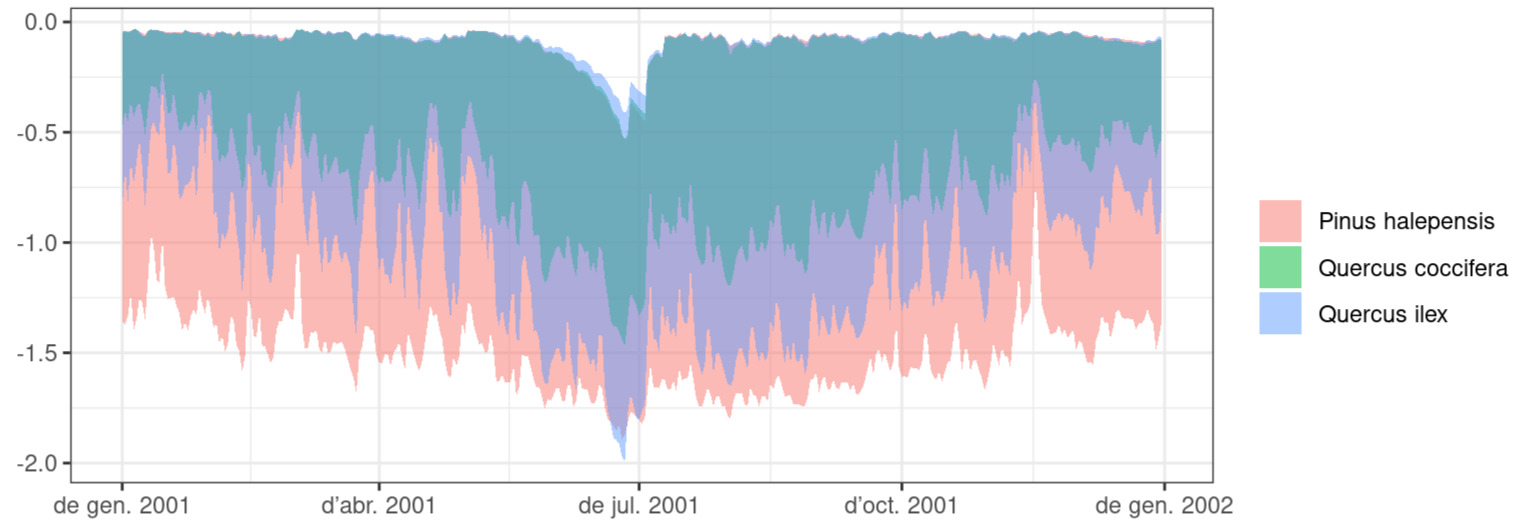
```r
S_adv <- spwb(x_adv, examplemeteo, latitude = 41.82592, elevation = 100)
```

Function `spwb()` returns a list of class `spwb`, like the basic water balance model, but which contains more information:

```
 [1] "latitude"      "topography"    "weather"       "spwbInput"    
 [5] "spwbOutput"    "WaterBalance"  "EnergyBalance" "Temperature"  
 [9] "Soil"          "Snow"          "Stand"         "Plants"       
[13] "SunlitLeaves"  "ShadeLeaves"   "subdaily"     
```

As before, post-processing of simulation results can be done using functions `summary()`, `extract()` or `plot()`:

```r
plot(S_adv, type="LeafPsiRange", bySpecies = TRUE)
```



Alternatively, one can interactively create plots using function `shinyplot()`, e.g.:

```r
shinyplot(S_adv)
```

5. Modifying model inputs

Modifying forest input data

Medfate uses allometric equations to estimate structural properties such as leaf area index (LAI) or the crown ratio (CR).

Let's imagine one is not happy with a particular cohort parameter. For example, LAI estimates produced by `spwbInput()` do not match known values:

```r
x_adv$above$LAI_live
```

```
[1] 0.84874773 0.70557382 0.03062604
```

One possibility is to specify LAI values directly in the `forest` object, as can be found in the example dataset:

```r
exampleforest2
```

```
$treeData
           Species  N DBH Height Z50  Z95 LAI CrownRatio
1 Pinus halepensis NA  NA    800 100  600 0.8       0.66
2     Quercus ilex NA  NA    660 300 1000 0.5       0.60

$shrubData
            Species Cover Height Z50  Z95  LAI CrownRatio
1 Quercus coccifera    NA     80 200 1000 0.03        0.8

$herbCover
[1] NA

$herbHeight
[1] 20

$herbLAI
[1] 0.25

$seedBank
[1] Species Percent
<0 files> (o «row.names» de longitud 0)

attr(,"class")
[1] "forest" "list"
```

Modifying species and cohort parameters

Species-level parameters

Advanced users may desire to have control on species-level parameter values used in simulation.

One can use function `modifySpParams()` to modify values in the species parameter table (we could also do it manually).

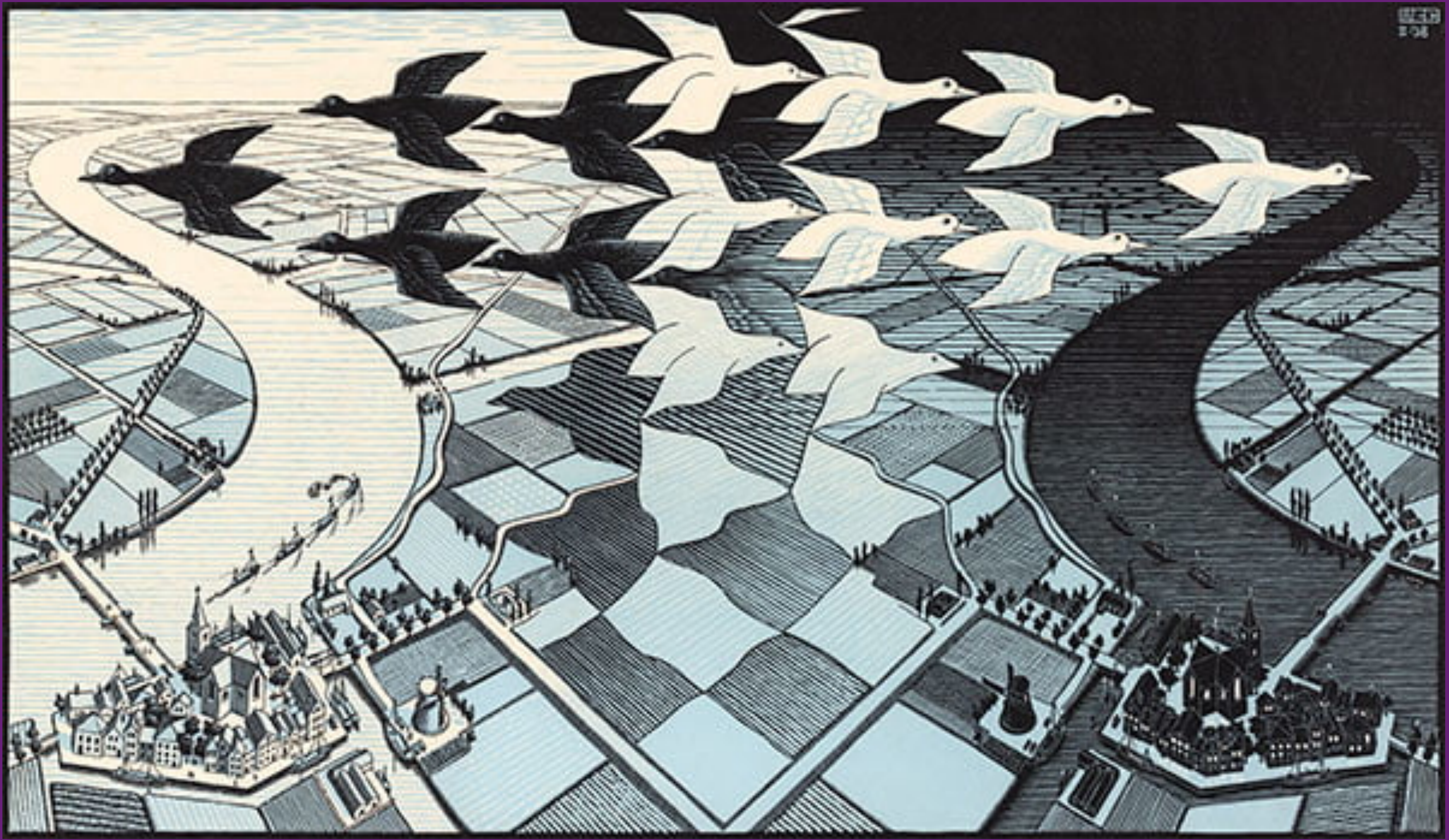
Cohort-level parameters

Cohort-level parameters may also be modified. However, one should not manually modify simulation input objects (e.g. `x_adv`) because some parameters are related and we may break their relationships.

Instead, function `modifyInputParams()` is recommended:

```r
x_mod <- modifyInputParams(x_adv, c("T2_168/VCstem_d" = -7.0))
```

which will display messages describing the parameters that are modified.


M.C. Escher - Night and day, 1938