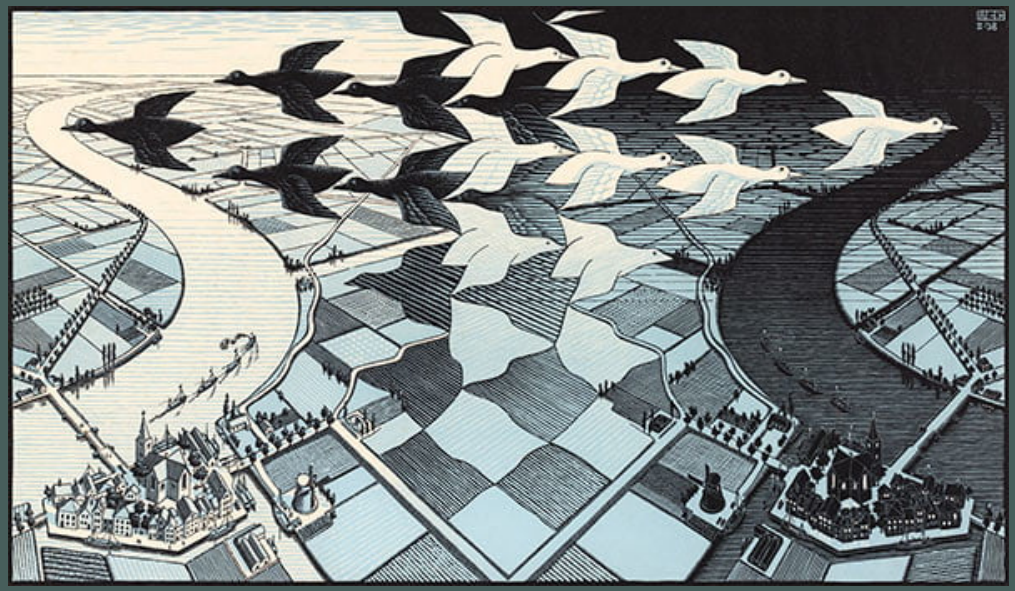
2.2 - Forest water/energy balance (practice)

Miquel De Cáceres, Victor Granda, Aitor Ameztegui

Ecosystem Modelling Facility

2022-06-14



Outline

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

1. Water balance input object

Creating the water balance input object

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

```
data(exampleforestMED)
data(SpParamsMED)
```

1. Water balance input object

Creating the water balance input object

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

```
data(exampleforestMED)
data(SpParamsMED)
```

a `soil` object:

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

1. Water balance input object

Creating the water balance input object

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

```
data(exampleforestMED)
data(SpParamsMED)
```

a `soil` object:

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

and a simulation control list:

```
control <- defaultControl("Granier")
```

1. Water balance input object

Creating the water balance input object

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

```
data(exampleforestMED)
data(SpParamsMED)
```

a `soil` object:

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

and a simulation control list:

```
control <- defaultControl("Granier")
```

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

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

1. Water balance input object

Structure of the water balance input object (1)

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

```
names(x)
```

```
##  [1] "control"             "soil"                "canopy"              "cohorts"
##  [5] "above"               "below"               "belowLayers"         "paramsPhenology"
##  [9] "paramsAnatomy"       "paramsInterception"  "paramsTranspiration" "paramsWaterStorage"
## [13] "internalPhenology"   "internalWater"
```

1. Water balance input object

Structure of the water balance input object (1)

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

```
names(x)
```

```
##  [1] "control"             "soil"                "canopy"              "cohorts"
##  [5] "above"               "below"               "belowLayers"         "paramsPhenology"
##  [9] "paramsAnatomy"       "paramsInterception"  "paramsTranspiration" "paramsWaterStorage"
## [13] "internalPhenology"   "internalWater"
```

Elements `soil` and `control` contain copies of the parameters used in the call to `forest2spwbInput()`.

1. Water balance input object

Structure of the water balance input object (1)

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

```
names(x)
```

```
##  [1] "control"             "soil"                "canopy"              "cohorts"
##  [5] "above"               "below"               "belowLayers"         "paramsPhenology"
##  [9] "paramsAnatomy"       "paramsInterception"  "paramsTranspiration" "paramsWaterStorage"
## [13] "internalPhenology"   "internalWater"
```

Elements `soil` and `control` contain copies of the parameters used in the call to `forest2spwbInput()`.

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

```
x$cohorts
```

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

1. Water balance input object

Structure of the water balance input object (2)

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

```
x$above
```

```
##          H        CR        N   LAI_live LAI_expanded LAI_dead
## T1_148 800 0.6605196 168.0000 0.96734365   0.96734365        0
## T2_168 660 0.6055642 384.0000 0.86167321   0.86167321        0
## S1_165  80 0.8032817 749.4923 0.03928201   0.03928201        0
```

1. Water balance input object

Structure of the water balance input object (2)

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

```
x$above
```

```
##          H        CR        N   LAI_live LAI_expanded LAI_dead
## T1_148 800 0.6605196 168.0000 0.96734365   0.96734365        0
## T2_168 660 0.6055642 384.0000 0.86167321   0.86167321        0
## S1_165  80 0.8032817 749.4923 0.03928201   0.03928201        0
```

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

```
x$below
```

```
##        Z50  Z95 fineRootBiomass coarseRootSoilVolume
## T1_148 100  600      1574.98321             4.381122
## T2_168 300 1000       667.64217             7.526346
## S1_165 200 1000        13.57757             5.544086
```

1. Water balance input object

Structure of the water balance input object (3)

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

```
x$paramsTranspiration
```

```
##             Gswmin  Tmax_LAI   Tmax_LAIsq Psi_Extract Psi_Critic      WUE WUE_decay
## T1_148 0.003086667 0.1751014 -0.007840361   -2.303000   -5.14000 6.107080 0.6831032
## T2_168 0.004473333 0.1123937 -0.005032554   -1.964000   -6.96500 7.564588 0.2387449
## S1_165 0.010455247 0.1340000 -0.006000000   -2.121073   -6.95092 5.000000 0.2812000
```

1. Water balance input object

Structure of the water balance input object (3)

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

```
x$paramsTranspiration
```

```
##             Gswmin  Tmax_LAI   Tmax_LAIsq Psi_Extract Psi_Critic      WUE WUE_decay
## T1_148 0.003086667 0.1751014 -0.007840361   -2.303000   -5.14000 6.107080 0.6831032
## T2_168 0.004473333 0.1123937 -0.005032554   -1.964000   -6.96500 7.564588 0.2387449
## S1_165 0.010455247 0.1340000 -0.006000000   -2.121073   -6.95092 5.000000 0.2812000
```

or …

```
x$paramsAnatomy
```

```
##           Al2As Ar2Al      SLA LeafDensity WoodDensity FineRootDensity      SRL RLD     r635
## T1_148 1317.523     1 5.140523   0.2982842   0.6077016       0.2982842 3172.572  10 1.964226
## T2_168 3908.823     1 6.340000   0.4893392   0.9008264       0.4893392 4398.812  10 1.805872
## S1_165 4189.325     1 4.980084   0.3709679   0.4389106       0.3709679 4398.812  10 2.289452
```

1. Water balance input object

Structure of the water balance input object (4)

Elements `internal*` contain cohort-level state variables, for example:

```
x$internalPhenology
```

```
##        gdd sen budFormation leafUnfolding leafSenescence leafDormancy phi
## T1_148   0   0        FALSE         FALSE          FALSE        FALSE   0
## T2_168   0   0        FALSE         FALSE          FALSE        FALSE   0
## S1_165   0   0        FALSE         FALSE          FALSE        FALSE   0
```

1. Water balance input object

Structure of the water balance input object (4)

Elements `internal*` contain cohort-level state variables, for example:

```
x$internalPhenology
```

```
##         gdd sen budFormation leafUnfolding leafSenescence leafDormancy phi
## T1_148    0   0         FALSE         FALSE          FALSE         FALSE   0
## T2_168    0   0         FALSE         FALSE          FALSE         FALSE   0
## S1_165    0   0         FALSE         FALSE          FALSE         FALSE   0
```

or...

```
x$internalWater
```

```
##         PlantPsi StemPLC
## T1_148    -0.033       0
## T2_168    -0.033       0
## S1_165    -0.033       0
```

2. Basic water balance

Water balance run

Let us assume we have an appropriate weather data frame:

```
data(examplemeteo)
```

2. Basic water balance

Water balance run

Let us assume we have an appropriate weather data frame:

```
data(examplemeteo)
```

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

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

```
## Initial soil water content (mm): 291.257
## Initial snowpack content (mm): 0
## Performing daily simulations
##
##  [Year 2001]:.................................
##
## Final soil water content (mm): 266.112
## Final snowpack content (mm): 0
## Change in soil water content (mm): -25.1443
## Soil water balance result (mm): -25.1443
## Change in snowpack water content (mm): 0
## Snowpack water balance result (mm): 0
## Water balance components:
##   Precipitation (mm) 513
##   Rain (mm) 462 Snow (mm) 51
##   Interception (mm) 96 Net rainfall (mm) 366
##   Infiltration (mm) 408 Runoff (mm) 9 Deep drainage (mm) 67
##   Soil evaporation (mm) 17 Transpiration (mm) 350
##   Plant extraction from soil (mm) 350  Plant water balance (mm) -0
```

2. Basic water balance

Water balance output object (1)

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

```
class(S)
```

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

2. Basic water balance

Water balance output object (1)

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

```
class(S)
```

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

It is interesting to inspect the list element names:

```
names(S)
```

```
##  [1] "latitude"     "topography"   "weather"      "spwbInput"    "spwbOutput"   "WaterBalance"
##  [7] "Soil"         "Stand"        "Plants"       "subdaily"
```

2. Basic water balance

Water balance output object (1)

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

```
class(S)
```

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

It is interesting to inspect the list element names:

```
names(S)
```

```
##  [1] "latitude"     "topography"   "weather"      "spwbInput"    "spwbOutput"   "WaterBalance"
##  [7] "Soil"         "Stand"        "Plants"       "subdaily"
```

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`, `Stand`, `Plants`	Daily outputs (days as rows, variables as columns)
`subdaily`	Sub-daily outputs (not relevant here)

2. Basic water balance

Water balance output object (2)

Actually, `Plants` is itself a list with several data frames of results by cohort (days as rows, cohorts as columns):

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

```
##  [1] "LAI"                 "LAIlive"             "FPAR"                "AbsorbedSWRFraction"
##  [5] "Transpiration"       "GrossPhotosynthesis" "PlantPsi"            "StemPLC"
##  [9] "PlantWaterBalance"   "LeafRWC"             "StemRWC"             "LFMC"
## [13] "PlantStress"
```

2. Basic water balance

Summaries

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 obtain the average soil moisture and water potentials by months one can use:

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

2. Basic water balance

Summaries

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 obtain the average soil moisture and water potentials by months one can use:

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

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:

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

2. Basic water balance

Summaries

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 obtain the average soil moisture and water potentials by months one can use:

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

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:

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

The `summary` function can be also used to aggregate the output by species. In this case, the values of plant cohorts belonging to the same species will be averaged using LAI values as weights:

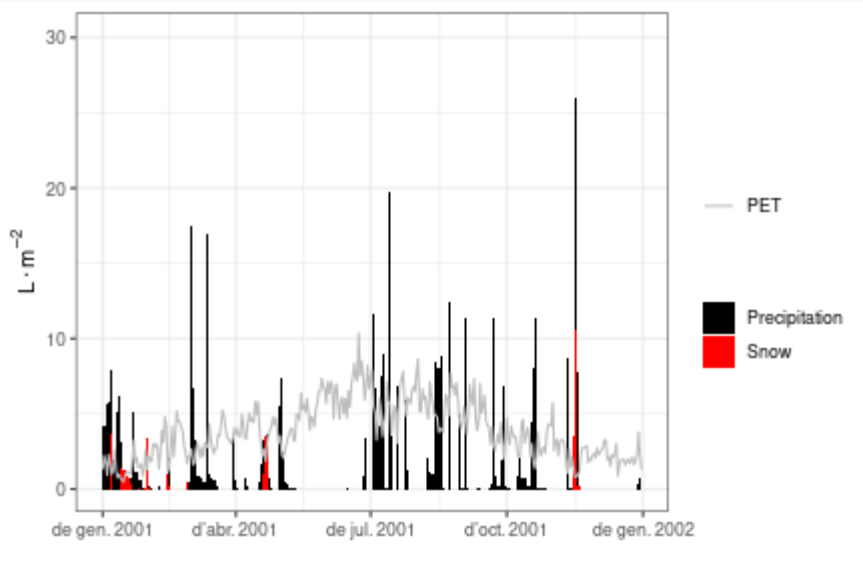
```
summary(S, freq="months", output="PlantStress", bySpecies = TRUE)
```

2. Basic water balance

Plots

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:

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

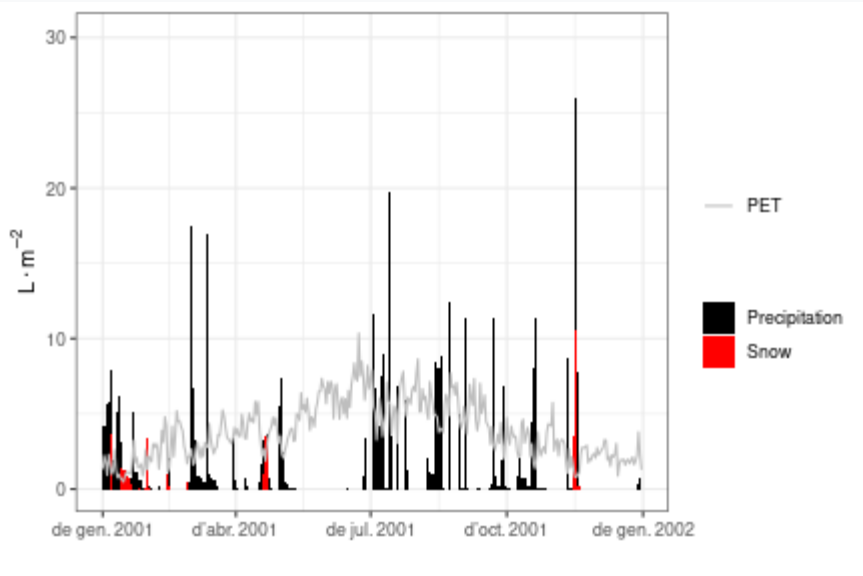


2. Basic water balance

Plots

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:

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



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

2. Basic water balance

Interactive plots

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

```
shinyplot(S)
```

2. Basic water balance

Post-processing functions

The package provides some functions to extract or transform specific outputs from `spwb()` simulations.

2. Basic water balance

Post-processing functions

The package provides some functions to extract or transform specific outputs from `spwb()` simulations.

Function `droughtStress()` allows calculating several plant stress indices, such as the maximum drought stress value per month:

```
DS <- droughtStress(S, index = "MDS", freq = "months", draw=FALSE)
head(DS)
```

```
##                   T1_148       T2_168       S1_165
## 2001-01-01 3.695504e-06 4.970814e-06 4.334085e-06
## 2001-02-01 2.784358e-05 2.912919e-05 2.932397e-05
## 2001-03-01 2.979376e-05 3.105366e-05 3.133203e-05
## 2001-04-01 1.100252e-04 1.100929e-04 1.138923e-04
## 2001-05-01 3.384258e-04 3.399830e-04 3.507640e-04
## 2001-06-01 9.831144e-03 9.765281e-03 1.014274e-02
```

2. Basic water balance

Post-processing functions

The package provides some functions to extract or transform specific outputs from `spwb()` simulations.

Function `droughtStress()` allows calculating several plant stress indices, such as the maximum drought stress value per month:

```
DS <- droughtStress(S, index = "MDS", freq = "months", draw=FALSE)
head(DS)
```

```
##                   T1_148       T2_168       S1_165
## 2001-01-01 3.695504e-06 4.970814e-06 4.334085e-06
## 2001-02-01 2.784358e-05 2.912919e-05 2.932397e-05
## 2001-03-01 2.979376e-05 3.105366e-05 3.133203e-05
## 2001-04-01 1.100252e-04 1.100929e-04 1.138923e-04
## 2001-05-01 3.384258e-04 3.399830e-04 3.507640e-04
## 2001-06-01 9.831144e-03 9.765281e-03 1.014274e-02
```

Other similar post-processing functions are `waterUseEfficiency()` or `fireHazard()`.

They (should) also work on the output of functions `growth()` and `fordyn()`.

3. Evaluating model performance

Evaluation metrics

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

3. Evaluating model performance

Evaluation metrics

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

For example, a single evaluation metric can be calculated:

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

```
## [1] 0.008413186
```

3. Evaluating model performance

Evaluation metrics

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

For example, a single evaluation metric can be calculated:

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

```
## [1] 0.008413186
```

or many of them:

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

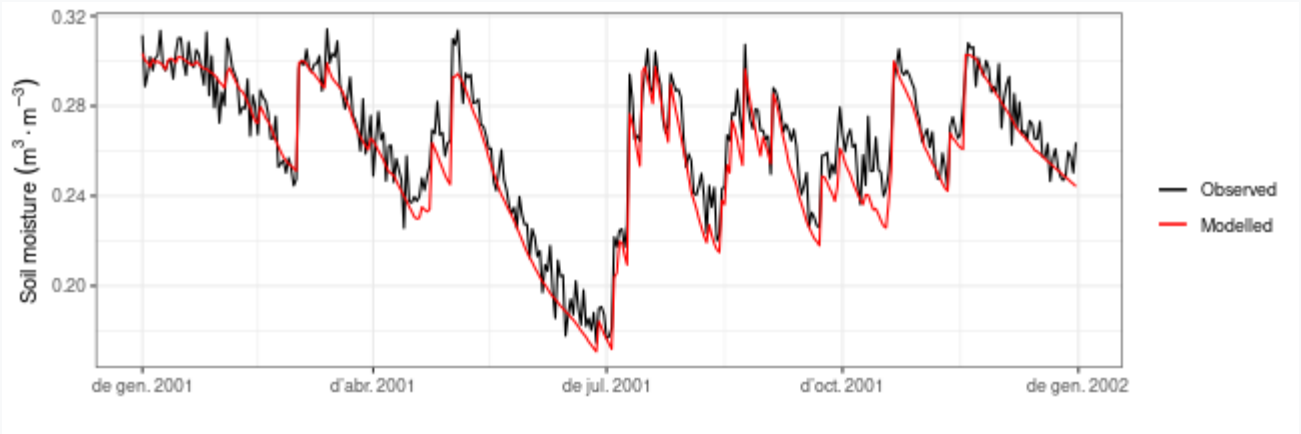
```
##             n          Bias      Bias.rel           MAE       MAE.rel             r           NSE
## 365.000000000 -0.006193322 -2.346516203 0.008413186 3.187574799 0.966748526 0.887956708
##       NSE.abs
##   0.649736575
```

3. Evaluating model performance

Evaluation plots and interactive evaluation

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

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



3. Evaluating model performance

Evaluation plots and interactive evaluation

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

```
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:

```
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:

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

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:

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

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

```
x_adv <- forest2spwbInput(exampleforestMED, examplesoil, SpParamsMED, control)
```

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:

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

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

```
x_adv <- forest2spwbInput(exampleforestMED, examplesoil, SpParamsMED, control)
```

The water balance input object contains the same elements...

```
names(x_adv)
```

```
##  [1] "control"             "soil"                "canopy"              "cohorts"
##  [5] "above"               "below"               "belowLayers"         "paramsPhenology"
##  [9] "paramsAnatomy"       "paramsInterception"  "paramsTranspiration" "paramsWaterStorage"
## [13] "internalPhenology"   "internalWater"
```

4. Advanced water/energy balance

Creating an input object for the advanced model

… but the main difference with the basic model is in the number of parameters, e.g.:

```
x_adv$paramsTranspiration
```

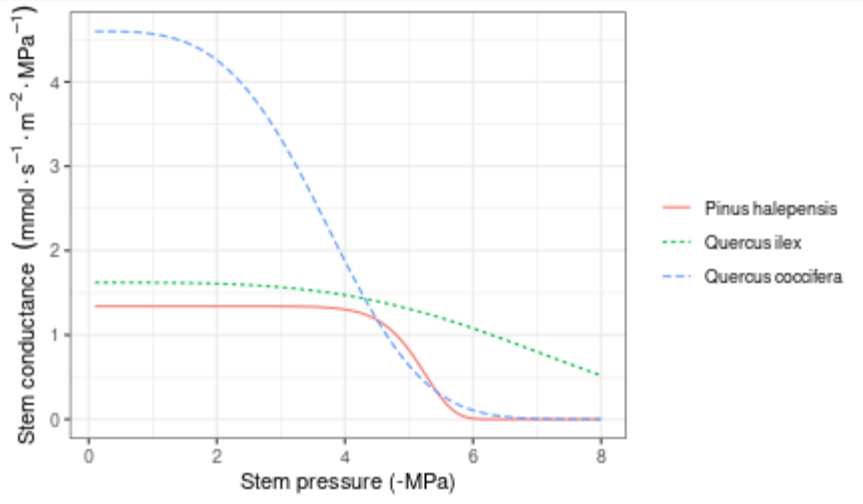
```
##             Gswmin    Gswmax  Vmax298  Jmax298 Kmax_stemxylem Kmax_rootxylem VCleaf_kmax  VCleaf_c
## T1_148 0.003086667 0.2850000 72.19617 124.1687           0.15           0.60    4.000000 11.137051
## T2_168 0.004473333 0.2007222 68.51600 118.7863           0.40           1.60    4.000000  1.339370
## S1_165 0.010455247 0.2830167 62.78100 118.4486           0.29           1.16    9.579077  1.844224
##         VCleaf_d VCstem_kmax  VCstem_c  VCstem_d VCroot_kmax  VCroot_c  VCroot_d VGrhizo_kmax
## T1_148 -2.380849    1.339563 12.710000 -5.290000    1.505250 11.137051 -3.065569     29078608
## T2_168 -2.582279    1.620936  3.560000 -7.720000    1.730249  1.339370 -2.214081     92609924
## S1_165 -3.030130    4.599269  3.537784 -4.126512    4.660990  1.760307 -2.797092    473105480
##        Plant_kmax
## T1_148  0.6021000
## T2_168  0.6920994
## S1_165  1.8643961
```

4. Advanced water/energy balance

Vulnerability curves

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:

```
hydraulics_vulnerabilityCurvePlot(x_adv, type="stem", speciesNames = TRUE)
```

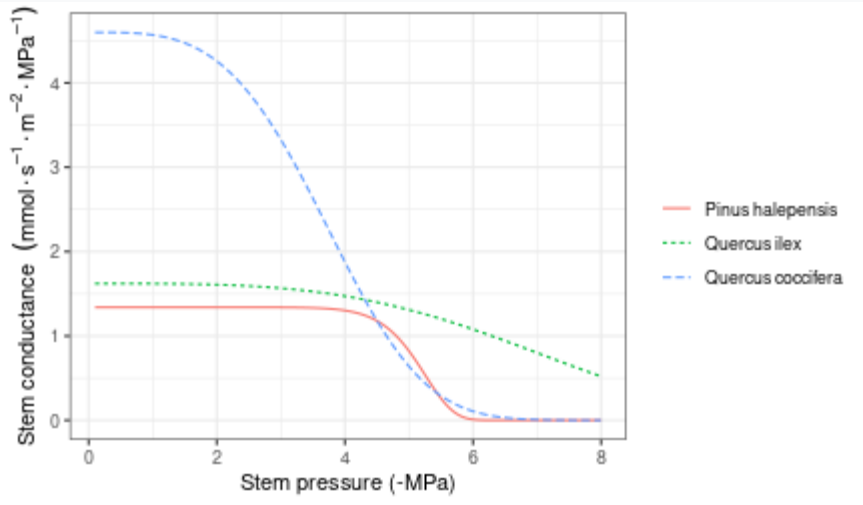


4. Advanced water/energy balance

Vulnerability curves

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:

```
hydraulics_vulnerabilityCurvePlot(x_adv, type="stem", speciesNames = TRUE)
```



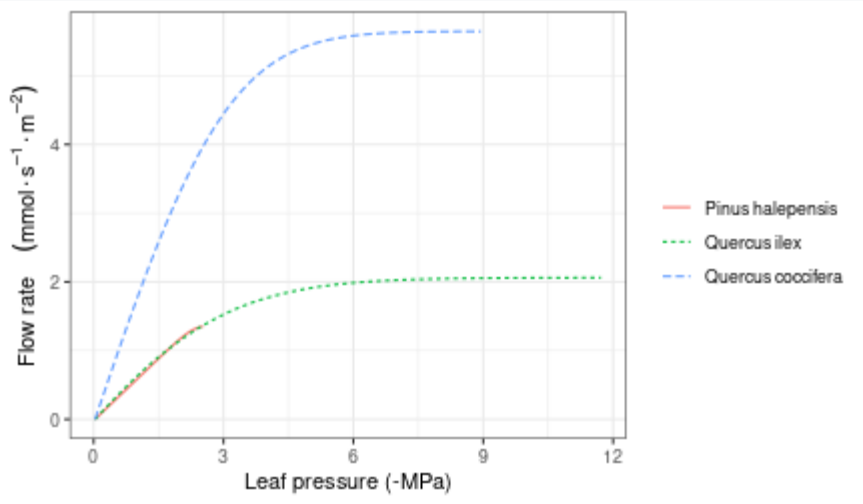
The maximum values and shape of vulnerability curves for leaves and stems are regulated by parameters in `paramsTranspiration`.

4. Advanced water/energy balance

Supply functions

The vulnerability curves conforming the hydraulic network are used in the model to build the **supply function**, which relates water flow (i.e. transpiration) with the drop of water potential along the whole hydraulic pathway.

```
hydraulics_supplyFunctionPlot(x_adv, examplesoil, type="E", speciesNames = TRUE)
```

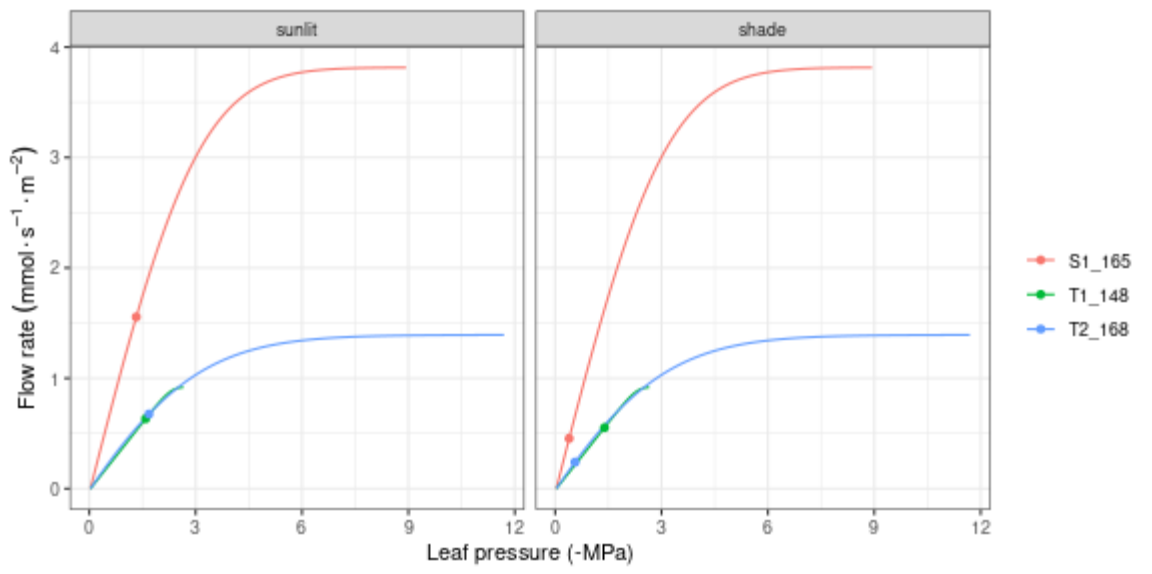


4. Advanced water/energy balance

Stomatal regulation

Stomatal conductance is determined after building a photosynthesis function corresponding to the supply function and finding the value of stomatal conductance that maximizes carbon revenue while avoiding hydraulic damage: the *profit-maximization* approach.

```
transp_stomatalRegulationPlot(x_adv, examplemeteo, day = 100, timestep=12,
                              latitude = 41.82592, elevation = 100, type="E")
```



4. Advanced water/energy balance

Water/energy balance run for a single day

Since the model operates at a daily and sub-daily temporal scales, it is possible to perform soil water balance for one day only, by using function `spwb_day()`:

```
d = 100
sd1<-spwb_day(x_adv, rownames(examplemeteo)[d],
              examplemeteo$MinTemperature[d], examplemeteo$MaxTemperature[d],
              examplemeteo$MinRelativeHumidity[d], examplemeteo$MaxRelativeHumidity[d],
              examplemeteo$Radiation[d], examplemeteo$WindSpeed[d],
              latitude = 41.82592, elevation = 100,
              slope= 0, aspect = 0, prec = examplemeteo$Precipitation[d])
```

4. Advanced water/energy balance

Water/energy balance run for a single day

Since the model operates at a daily and sub-daily temporal scales, it is possible to perform soil water balance for one day only, by using function `spwb_day()`:

```
d = 100
sd1<-spwb_day(x_adv, rownames(examplemeteo)[d],
              examplemeteo$MinTemperature[d], examplemeteo$MaxTemperature[d],
              examplemeteo$MinRelativeHumidity[d], examplemeteo$MaxRelativeHumidity[d],
              examplemeteo$Radiation[d], examplemeteo$WindSpeed[d],
              latitude = 41.82592, elevation = 100,
              slope= 0, aspect = 0, prec = examplemeteo$Precipitation[d])
```

The output of `spwb_day()` is a list with several elements:

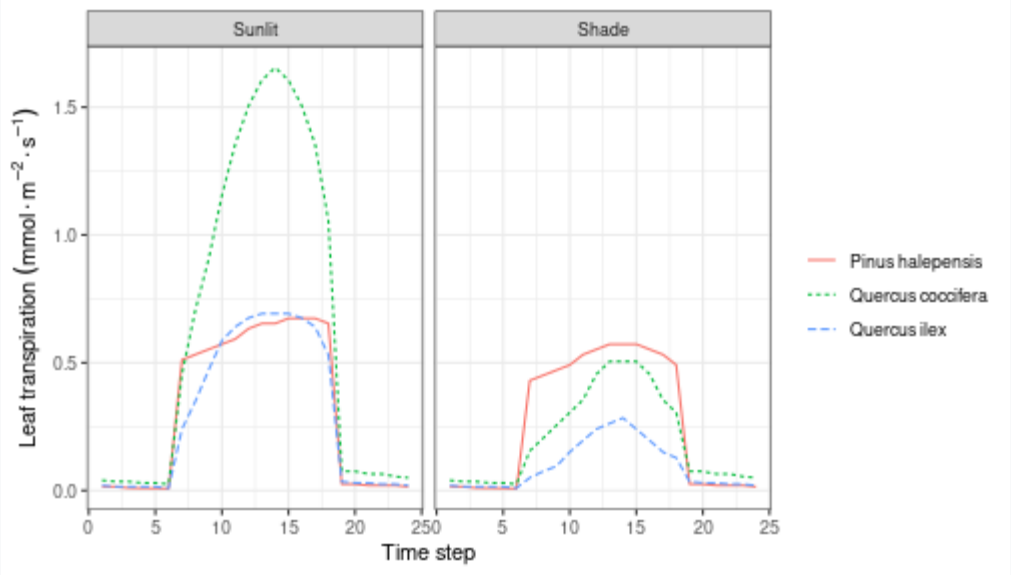
```
names(sd1)
```

```
##  [1] "cohorts"          "topography"       "weather"          "WaterBalance"     "EnergyBalance"
##  [6] "Soil"             "Stand"            "Plants"           "RhizoPsi"         "SunlitLeaves"
## [11] "ShadeLeaves"      "ExtractionInst"   "PlantsInst"       "SunlitLeavesInst" "ShadeLeavesInst"
## [16] "LightExtinction"  "LWRExtinction"    "CanopyTurbulence"
```

4. Advanced water/energy balance

Plotting single-day results

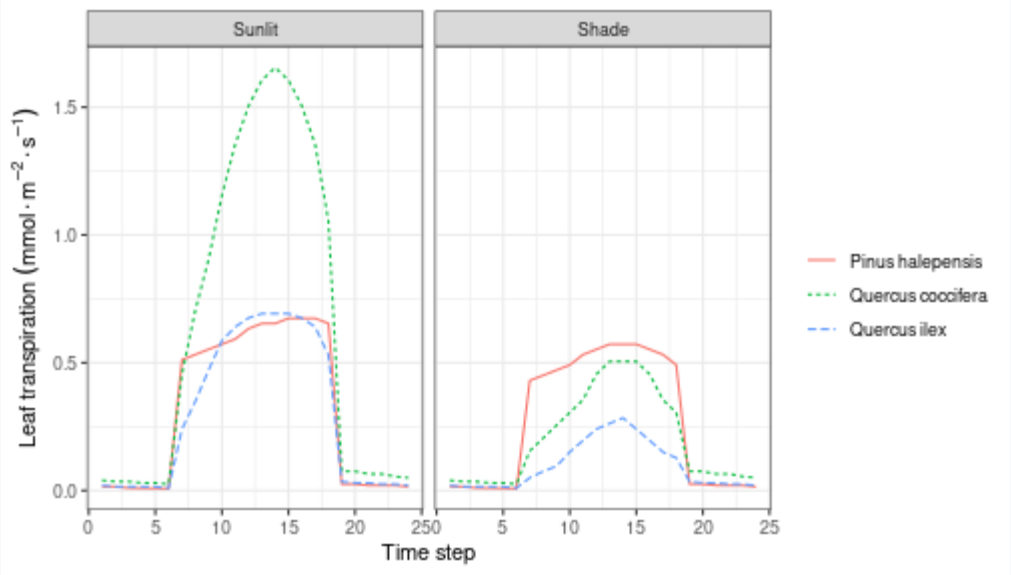
```
plot(sd1, type = "LeafTranspiration", bySpecies = TRUE)
```



4. Advanced water/energy balance

Plotting single-day results

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



More conveniently, you can examine multiple plots interactively:

```
shinyplot(sd1)
```

4. Advanced water/energy balance

Resetting the input object

However, running `spwb_day()` modified the input object. In particular, the soil moisture at the end of the simulation was:

```
x_adv$soil$W
```

```
## [1] 0.9926198 0.9982895 0.9997181 0.9998917
```

4. Advanced water/energy balance

Resetting the input object

However, running `spwb_day()` modified the input object. In particular, the soil moisture at the end of the simulation was:

```
x_adv$soil$W
```

```
## [1] 0.9926198 0.9982895 0.9997181 0.9998917
```

We simply use function `resetInputs()` to reset state variables to their default values, so that the new simulation is not affected by the end state of the previous simulation:

```
resetInputs(x_adv)
x_adv$soil$W
```

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

4. Advanced water/energy balance

Water/energy balance run for multiple days

We can now run the advanced water balance model (which takes 1 min aprox.)

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

4. Advanced water/energy balance

Water/energy balance run for multiple days

We can now run the advanced water balance model (which takes 1 min aprox.)

```
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:

```
names(S_adv)
```

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

4. Advanced water/energy balance

Summaries, plots and interactive plots

Summaries and plots can be obtained from simulation results, using functions `summary()`:

```
summary(S_adv, freq="months", output="PlantStress", bySpecies = TRUE)
```

4. Advanced water/energy balance

Summaries, plots and interactive plots

Summaries and plots can be obtained from simulation results, using functions `summary()`:

```
summary(S_adv, freq="months", output="PlantStress", bySpecies = TRUE)
```

and *plot*():

```
plot(S_adv, type="LeafPsiMin", bySpecies = TRUE)
```

4. Advanced water/energy balance

Summaries, plots and interactive plots

Summaries and plots can be obtained from simulation results, using functions `summary()`:

```
summary(S_adv, freq="months", output="PlantStress", bySpecies = TRUE)
```

and $plot()$:

```
plot(S_adv, type="LeafPsiMin", bySpecies = TRUE)
```

Alternatively, one can interactively create plots using function $sh \in yplot()$, e.g.:

```
shinyplot(S_adv)
```

5. Modifying model inputs

Let's imagine one is not happy with a particular cohort parameter. For example, LAI estimates produced by `forest2spwbInput()` do not match known values:

```
x_adv$above
```

```
##          H        CR        N   LAI_live LAI_expanded LAI_dead
## T1_148 800 0.6605196 168.0000 0.96734365   0.96734365        0
## T2_168 660 0.6055642 384.0000 0.86167321   0.86167321        0
## S1_165  80 0.8032817 749.4923 0.03928201   0.03928201        0
```

5. Modifying model inputs

Let's imagine one is not happy with a particular cohort parameter. For example, LAI estimates produced by `forest2spwbInput()` do not match known values:

```
x_adv$above
```

```
##          H        CR        N   LAI_live LAI_expanded LAI_dead
## T1_148 800 0.6605196 168.0000 0.96734365   0.96734365        0
## T2_168 660 0.6055642 384.0000 0.86167321   0.86167321        0
## S1_165  80 0.8032817 749.4923 0.03928201   0.03928201        0
```

We should not manually modify `x_adv` because some parameters are related and we may break their relationship.

5. Modifying model inputs

Let's imagine one is not happy with a particular cohort parameter. For example, LAI estimates produced by `forest2spwbInput()` do not match known values:

```
x_adv$above
```

```
##          H        CR        N   LAI_live LAI_expanded LAI_dead
## T1_148 800 0.6605196 168.0000 0.96734365   0.96734365        0
## T2_168 660 0.6055642 384.0000 0.86167321   0.86167321        0
## S1_165  80 0.8032817 749.4923 0.03928201   0.03928201        0
```

We should not manually modify `x_adv` because some parameters are related and we may break their relationship.

Instead, we should use function `modifyInputParams()`:

```
x_mod <- modifyInputParams(x_adv, c("T2_168/LAI_live" = 1.1))
```

which will display messages describing the parameters that are modified.

M.C. Escher - Night and day, 1938