Forest growth and dynamics (practice)

Miquel De Cáceres, Rodrigo Balaguer
Ecosystem Modelling Facility, CREAF

Outline

1. Forest growth inputs
2. Running forest growth
3. Evaluation of growth predictions
4. Forest dynamics
5. Forest dynamics including management

M.C. Escher - Up and down, 1947

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```r
data(exampleforest)
```

a species parameter data frame:

```r
data(SpParamsMED)
```

a soil data frame:

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

and simulation control list:

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

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

```r
x <- forest2growthInput(exampleforest, examplesoil, SpParamsMED, control)
```

Structure of the growth input object (1)

The growth input object is a `list` with several elements:

```
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"          "paramsGrowth"               
[17] "paramsMortalityRegeneration" "paramsAllometries"          
[19] "internalPhenology"           "internalWater"              
[21] "internalLAIDistribution"     "internalCarbon"             
[23] "internalAllocation"          "internalMortality"          
[25] "internalFCCS"               
```

Element `above` contains the above-ground structure data that we already know, but with an additional column `SA` that describes the estimated initial amount of *sapwood area*:

```
       SP        N   DBH Cover   H        CR          SA   LAI_live
T1_148 148 168.0000 37.55    NA 800 0.6605196 383.4520992 0.84874773
T2_168 168 384.0000 14.60    NA 660 0.6055642  47.0072886 0.70557382
S1_165 165 749.4923    NA  3.75  80 0.8032817   0.9753929 0.03062604
       LAI_expanded LAI_dead LAI_nocomp    Loading ObsID
T1_148   0.84874773        0 1.29720268 0.32447403  <NA>
T2_168   0.70557382        0 1.01943205 0.20102636  <NA>
S1_165   0.03062604        0 0.04412896 0.01407945  <NA>
```

Structure of the growth input object (2)

Elements starting with `params*` contain cohort-specific model parameters. An important set of parameters are in `paramsGrowth`:

```
          RERleaf RERsapwood  RERfineroot CCleaf CCsapwood CCfineroot
T1_148 0.01210607   5.15e-05 0.0009610199 1.5905      1.47        1.3
T2_168 0.01757808   5.15e-05 0.0072846640 1.4300      1.47        1.3
S1_165 0.02647746   5.15e-05 0.0072846640 1.5320      1.47        1.3
       RGRleafmax RGRsapwoodmax RGRcambiummax RGRfinerootmax SRsapwood
T1_148       0.09            NA   0.002628095            0.1  0.000135
T2_168       0.09            NA   0.002500000            0.1  0.000135
S1_165       0.09         0.002            NA            0.1  0.000135
        SRfineroot      RSSG fHDmin fHDmax     WoodC
T1_148 0.001897231 0.3725000     80    160 0.4979943
T2_168 0.001897231 0.9500000     40    100 0.4740096
S1_165 0.001897231 0.7804035     NA     NA 0.4749178
```

Elements starting with `internal*` contain state variables required to keep track of plant status. For example, the metabolic and storage carbon levels can be seen in `internalCarbon`:

```
       sugarLeaf starchLeaf sugarSapwood starchSapwood
T1_148 0.4029239 0.00925123    0.5738487      3.201897
T2_168 0.3585751 0.00925123    1.0741383      3.100817
S1_165 0.7223526 0.00925123    0.2857655      2.654773
```

2. Forest growth

Forest growth run

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

```r
G <- growth(x, examplemeteo, latitude = 41.82592, elevation = 100)
```

```
Initial plant cohort biomass (g/m2): 5068.34
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 biomass (g/m2): 5256.53
Change in plant biomass (g/m2): 188.193
Plant biomass balance result (g/m2): 188.193
Plant biomass balance components:
  Structural balance (g/m2) 104 Labile balance (g/m2) 92
  Plant individual balance (g/m2) 196 Mortality loss (g/m2) 8
Final plant water content (mm): 4.73794
Final soil water content (mm): 274.787
Final snowpack content (mm): 0
Change in plant water content (mm): 0.00793033
Plant water balance result (mm): -0.00116903
Change in soil water content (mm): -16.0878
Soil water balance result (mm): -16.0878
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
```

Growth output object

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

```
class(G)
```

```
[1] "growth" "list"
```

… whose elements are:

```
names(G)
```

```
 [1] "latitude"           "topography"         "weather"           
 [4] "growthInput"        "growthOutput"       "WaterBalance"      
 [7] "CarbonBalance"      "BiomassBalance"     "Soil"              
[10] "Snow"               "Stand"              "Plants"            
[13] "LabileCarbonBalance" "PlantBiomassBalance" "PlantStructure"    
[16] "GrowthMortality"   
```

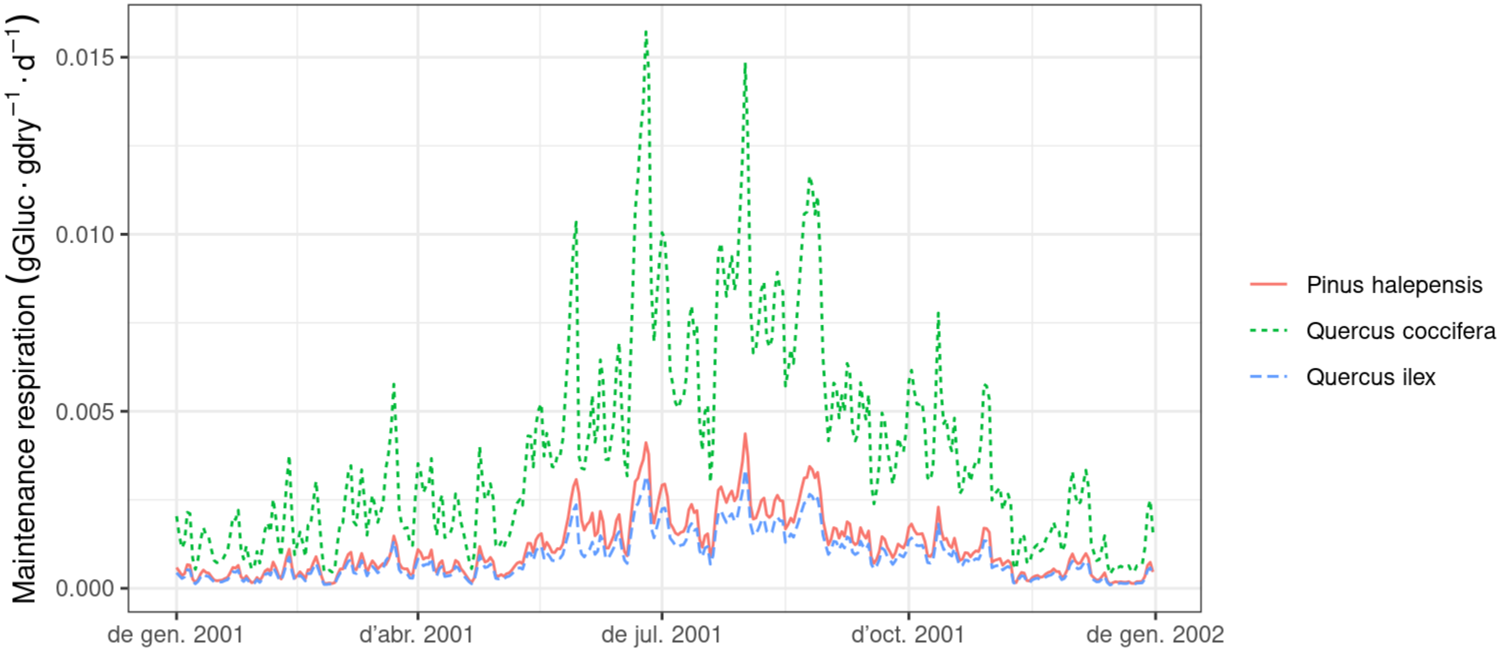
Elements	Information
`latitude`, `topography`, `weather`, `growthInput`	Copies of the information used in the call to `growth()`
`growthOutput`	State variables at the end of the simulation (can be used as input to a subsequent one)
`WaterBalance`, `Soil`, `Snow`, `Stand`, `Plants`	[same as `spwb` …]
`CarbonBalance`	Stand-level carbon blance
`LabileCarbonBalance`	Components of the individual-level labile carbon balance
`PlantBiomassBalance`	Components of indvidual- and cohort-level biomass balance
`PlantStructure`	Structural variables (DBH, height, sapwood area…)
`GrowthMortality`	Growth and mortality rates

Post-processing

Users can inspect the output of `growth()` simulations using functions `extract()`, `summary()` and `plot()` on the simulation output.

Several new plots are available in addition to those available for `spwb()` simulations (see `?plot.growth`). For example:

```r
plot(G, "MaintenanceRespiration", bySpecies = TRUE)
```



… but instead of typing all plots, we can call the interactive plot function `shinyplot()`.

3. Evaluation of growth predictions

Observed data frame

Evaluation of growth simulations will normally imply the comparison of predicted vs observed **basal area increment** (BAI) or **diameter increment** (DI) at a given temporal resolution.

Here, we illustrate the evaluation functions included in the package using a fake data set at *daily* resolution, consisting on the predicted values and some added error.

```
data(exampleobs)
head(exampleobs)
```

```
       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
```

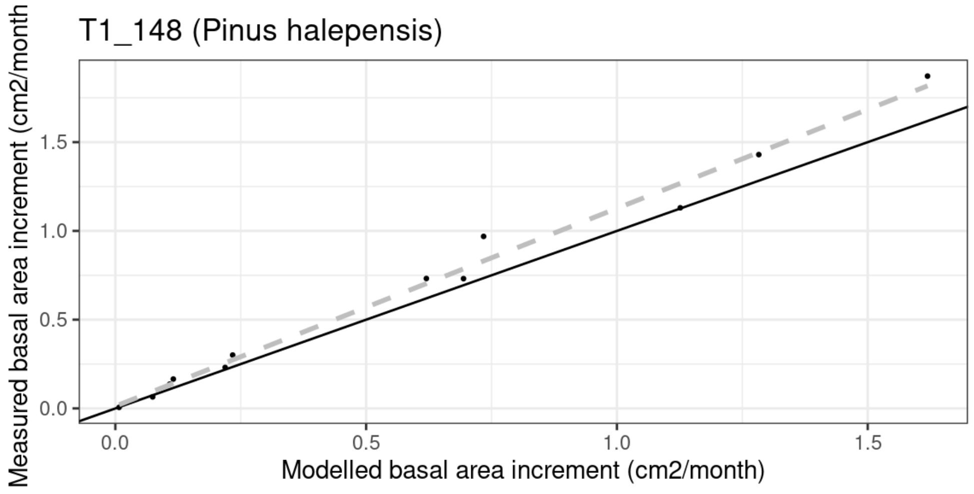
To specify observed growth data at *monthly* or *annual scale*, you should specify the first day of each month/year (e.g. `2001-01-01`, `2002-01-01`, etc for years) as row names in your observed data frame.

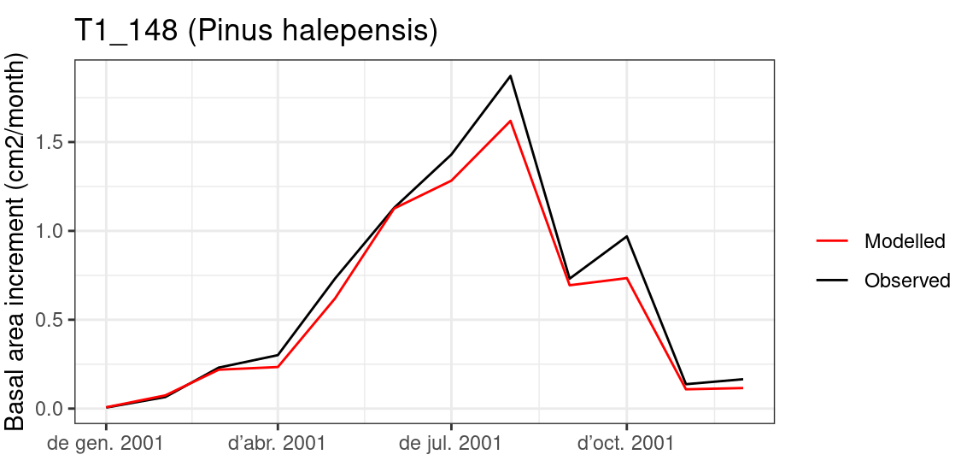
Evaluation plot

Assuming we want to evaluate the predictive capacity of the model in terms of monthly basal area increment for the *pine cohort* (i.e. `T1_148`), we can plot the relationship between observed and predicted values using `evaluation_plot()`:

```
evaluation_plot(G, exampleobs, "BAI",
                cohort = "T1_148",
                temporalResolution = "month",
                plotType = "scatter")
```

```
evaluation_plot(G, exampleobs, "BAI",
                cohort = "T1_148",
                temporalResolution = "month",
                plotType = "dynamics")
```





Using `temporalResolution = "month"` we indicate that simulated and observed data should be temporally aggregated to conduct the comparison.

The following code would help us quantifying the *strength* of the relationship:

```
evaluation_stats(G, exampleobs, "BAI", cohort = "T1_148", temporalResolution = "month")
```

```
          n        Bias    Bias.rel         MAE     MAE.rel           r 
12.00000000 -0.07781339 -12.01831387  0.07962106 12.29751007  0.99395791 
        NSE     NSE.abs 
 0.95935190  0.83966426 
```

4. Forest dynamics

Forest dynamics run

Weather preparation

In this vignette we will fake a three-year weather input by repeating the example weather data frame four times:

```
meteo <- rbind(examplemeteo, examplemeteo, examplemeteo, examplemeteo)
```

we need to update the dates in row names so that they span four consecutive years:

```
meteo$dates <- seq(as.Date("2001-01-01"),
                   as.Date("2004-12-30"), by="day")
```

Simulation

The call to `fordyn()` has the following form:

```
fd<-fordyn(exampleforest, examplesoil, SpParamsMED, meteo, control,
           latitude = 41.82592, elevation = 100)
```

```
Simulating year 2001 (1/4):  (a) Growth/mortality, (b) Regeneration nT = 2 nS = 1
Simulating year 2002 (2/4):  (a) Growth/mortality, (b) Regeneration nT = 2 nS = 1
Simulating year 2003 (3/4):  (a) Growth/mortality, (b) Regeneration nT = 2 nS = 1
Simulating year 2004 (4/4):  (a) Growth/mortality, (b) Regeneration nT = 2 nS = 1
```

Important

- `fordyn()` operates on `forest` objects directly, instead of using an intermediary object (such as `spwbInput` and `growthInput`).
- `fordyn()` calls function `growth()` internally for each simulated year, but all console output from `growth()` is hidden.

Forest dynamics output (1)

As with other models, the output of `fordyn()` is a list, which has the following elements:

Elements	Information
`StandSummary`, `SpeciesSummary`, `CohortSummary`	*Annual* summary statistics at different levels
`TreeTable`, `ShrubTable`	Structural variables of **living** cohorts at each annual time step.
`DeadTreeTable`, `DeadShrubTable`	Structural variables of **dead** cohorts at each annual time step
`CutTreeTable`, `CutShrubTable`	Structural variables of **cut** cohorts at each annual time step
`ForestStructures`	Vector of `forest` objects at each time step.
`GrowthResults`	Result of internally calling `growth()` at each time step.
`ManagementArgs`	Management arguments for a subsequent call to `fordyn()`.
`NextInputObject`, `NextForestObject`	Objects `growthInput` and `forest` to be used in a subsequent call to `fordyn()`.

Forest dynamics output (2)

For example, we can compare the initial `forest` object with the final one:

```r
exampleforest
```

```
$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

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

$herbCover
[1] 10

$herbHeight
[1] 20

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

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

```r
fd$NextForestObject
```

```
$treeData
           Species      DBH   Height        N Z50  Z95
1 Pinus halepensis 38.01410 824.7217 166.7796 100  600
2     Quercus ilex 15.04679 673.8471 382.6513 300 1000

$shrubData
            Species  Height    Cover Z50  Z95
1 Quercus coccifera 75.4552 3.237287 200 1000

$herbCover
[1] 10

$herbHeight
[1] 20

$seedBank
            Species Percent
1  Pinus halepensis     100
2      Quercus ilex     100
3 Quercus coccifera     100

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

Forest dynamics output (3)

The output includes **summary statistics** that describe the structural and compositional state of the forest corresponding to *each annual time step*.

For example, we can access *stand-level* statistics using:

```
fd$StandSummary
```

```
  Step NumTreeSpecies NumTreeCohorts NumShrubSpecies NumShrubCohorts
1    0              2              2               1               1
2    1              2              2               1               1
3    2              2              2               1               1
4    3              2              2               1               1
5    4              2              2               1               1
  TreeDensityLive TreeBasalAreaLive DominantTreeHeight DominantTreeDiameter
1        552.0000          25.03330           800.0000             37.55000
2        551.3663          25.20814           806.2122             37.66571
3        550.7269          25.38313           812.4144             37.78185
4        550.0818          25.55825           818.5877             37.89804
5        549.4309          25.73303           824.7217             38.01410
  QuadraticMeanTreeDiameter HartBeckingIndex ShrubCoverLive BasalAreaDead
1                  24.02949         53.20353       3.750000    0.00000000
2                  24.12711         52.82391       3.092051    0.03917375
3                  24.22476         52.45105       3.139858    0.03983747
4                  24.32243         52.08602       3.188231    0.04050957
5                  24.41991         51.72923       3.237287    0.04118879
  ShrubCoverDead BasalAreaCut ShrubCoverCut
1    0.000000000            0             0
2    0.005308898            0             0
3    0.004784468            0             0
4    0.004858342            0             0
5    0.004933117            0             0
```

Forest dynamics output (4)

Another useful output of `fordyn()` are tables in long format with cohort structural information (i.e. DBH, height, density, etc) for each time step:

```
fd$TreeTable
```

```
   Step Year Cohort           Species      DBH   Height        N Z50  Z95 ObsID
1     0   NA T1_148 Pinus halepensis 37.55000 800.0000 168.0000 100  600  <NA>
2     0   NA T2_168      Quercus ilex 14.60000 660.0000 384.0000 300 1000  <NA>
3     1 2001 T1_148 Pinus halepensis 37.66571 806.2122 167.6992 100  600  <NA>
4     1 2001 T2_168      Quercus ilex 14.71218 663.4915 383.6671 300 1000  <NA>
5     2 2002 T1_148 Pinus halepensis 37.78185 812.4144 167.3956 100  600  <NA>
6     2 2002 T2_168      Quercus ilex 14.82389 666.9588 383.3314 300 1000  <NA>
7     3 2003 T1_148 Pinus halepensis 37.89804 818.5877 167.0890 100  600  <NA>
8     3 2003 T2_168      Quercus ilex 14.93552 670.4135 382.9928 300 1000  <NA>
9     4 2004 T1_148 Pinus halepensis 38.01410 824.7217 166.7796 100  600  <NA>
10    4 2004 T2_168      Quercus ilex 15.04679 673.8471 382.6513 300 1000  <NA>
```

Note

The `NA` values in `Year` correspond to the initial state.

Forest dynamics output (5)

The same information can be shown for trees that are predicted to die during each simulated year:

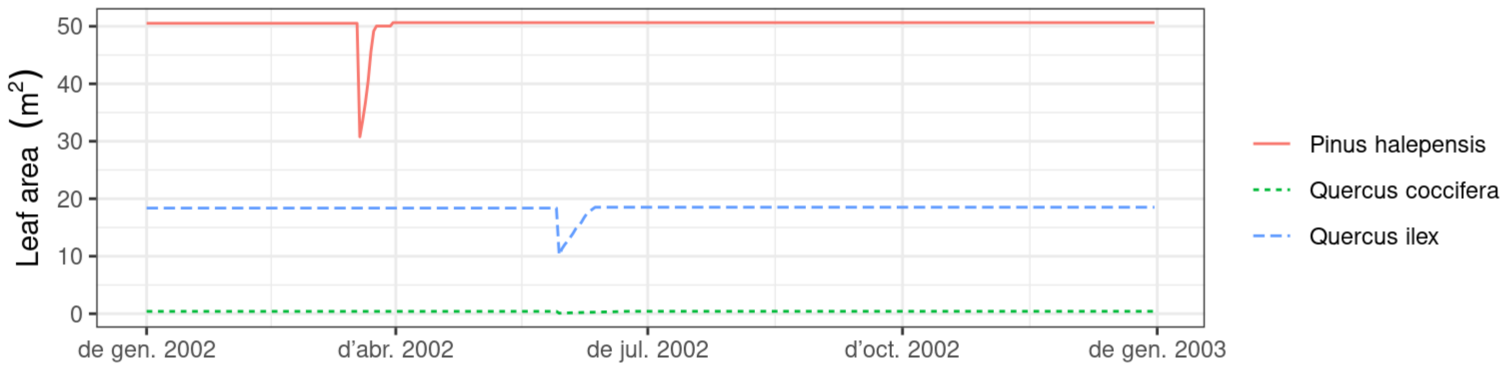
```
fd$DeadTreeTable
```

```
  Step Year Cohort          Species      DBH   Height         N N_starvation
1    1 2001 T1_148 Pinus halepensis 37.66571 806.2122 0.3007828            0
2    1 2001 T2_168      Quercus ilex 14.71218 663.4915 0.3328893            0
3    2 2002 T1_148 Pinus halepensis 37.78185 812.4144 0.3036481            0
4    2 2002 T2_168      Quercus ilex 14.82389 666.9588 0.3357419            0
5    3 2003 T1_148 Pinus halepensis 37.89804 818.5877 0.3065253            0
6    3 2003 T2_168      Quercus ilex 14.93552 670.4135 0.3386082            0
7    4 2004 T1_148 Pinus halepensis 38.01410 824.7217 0.3094089            0
8    4 2004 T2_168      Quercus ilex 15.04679 673.8471 0.3414828            0
  N_dessication N_burnt Z50  Z95 ObsID
1             0       0 100  600  <NA>
2             0       0 300 1000  <NA>
3             0       0 100  600  <NA>
4             0       0 300 1000  <NA>
5             0       0 100  600  <NA>
6             0       0 300 1000  <NA>
7             0       0 100  600  <NA>
8             0       0 300 1000  <NA>
```

Post-processing

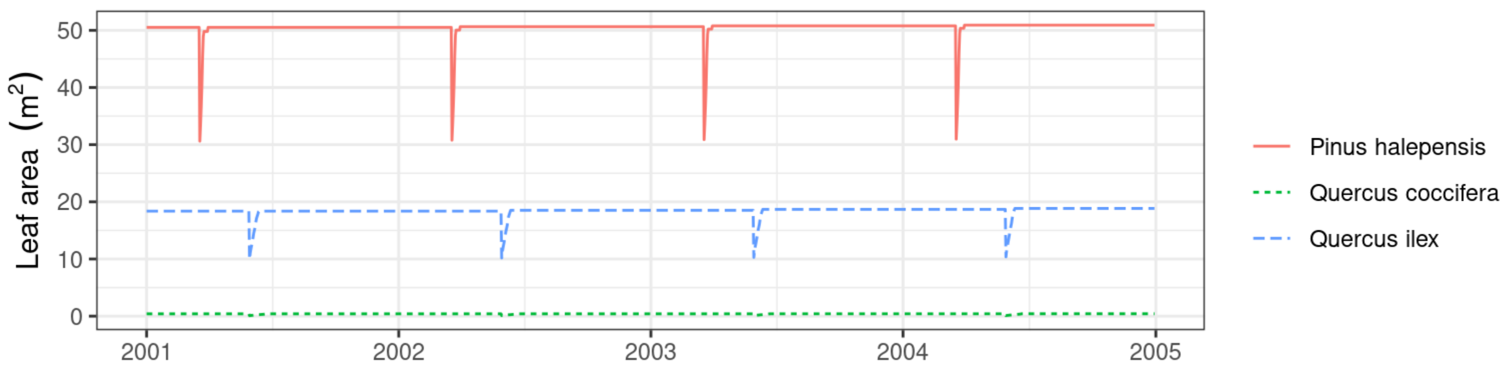
Accessing elements of `GrowthResults`, we can extract, summarize or plot simulation results for a particular year:

```
plot(fd$GrowthResults[[2]], "LeafArea", bySpecies = TRUE)
```



It is also possible to plot the whole series of results by passing a `fordyn` object to the `plot()` function:

```
plot(fd, "LeafArea", bySpecies = TRUE)
```



Finally, we can create interactive plots using function `shinyplot()`, in the same way as with other simulations.

5. Forest dynamics including management

Running simulations with management

`fordyn()` allows the user to supply an *arbitrary* function implementing a desired management strategy for the stand whose dynamics are to be simulated.

The package includes an in-built default function called `defaultManagementFunction()` along management parameter defaults provided by function `defaultManagementArguments()`.

To run simulations with management we need to define (and modify) management arguments…

```
# Default arguments
args <- defaultManagementArguments()
# Here one can modify defaults before calling fordyn()
#
```

… and call `fordyn()` specifying the management function and its arguments:

```
fd<-fordyn(exampleforest, examplesoil, SpParamsMED, meteo, control,
           latitude = 41.82592, elevation = 100,
           management_function = defaultManagementFunction,
           management_args = args)
```

When management is included, tables `CutTreeTable` and `CutShrubTable` will contain extraction data:

```
fd$CutTreeTable
fd$CutShrubTable
```

Forest management parameters (1)

Function `defaultManagementArguments()` returns a list with default values for *management parameters* to be used in conjunction with `defaultManagementFunction()`:

Element	Description
`type`	Management model, either 'regular' or 'irregular'
`targetTreeSpecies`	Either `"all"` for unspecific cuttings or a numeric vector of target tree species to be selected for cutting operations
`thinning`	Kind of thinning to be applied in irregular models or in regular models before the final cuts. Options are `"below"`, `"above"`, `"systematic"`, `"below-systematic"`, `"above-systematic"` or a string with the proportion of cuts to be applied to different diameter sizes
`thinningMetric`	The stand-level metric used to decide whether thinning is applied, either `"BA"` (basal area), `"N"` (density) or `"HB"` (Hart-Becking index)
`thinningThreshold`	The threshold value of the stand-level metric causing the thinning decision
`thinningPerc`	Percentage of stand's basal area to be removed in thinning operations
`minThinningInterval`	Minimum number of years between thinning operations
`finalMeanDBH`	Mean DBH threshold to start final cuts
`finalPerc`	String with percentages of basal area to be removed in final cuts, separated by '-' (e.g. "40-60-100")
`finalYearsBetweenCuts`	Number of years separating final cuts

Forest management parameters (2)

The same list includes *state variables* for management (these are modified during the simulation):

Element	Description
`yearsSinceThinning`	State variable to count the years since the last thinning ocurred
`finalPreviousStage`	Integer state variable to store the stage of final cuts ('0' before starting final cuts)
`finalYearsToCut`	State variable to count the years to be passed before new final cut is applied.

Tip

Instead of using the in-built management function, you could code your own management function and specify its own set of parameters!

M.C. Escher - Up and down, 1947