3.2 - Forest growth/dynamics (practice)

Miquel De Cáceres, Victor Granda

Ecosystem Modelling Facility

2022-12-02

Outline

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

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```
data(exampleforestMED)
```

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```
data(exampleforestMED)
```

a species parameter data frame:

```
data(SpParamsMED)
```

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```
data(exampleforestMED)
```

a species parameter data frame:

```
data(SpParamsMED)
```

a soil input object:

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

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```
data(exampleforestMED)
```

a species parameter data frame:

```
data(SpParamsMED)
```

a soil input object:

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

and simulation control list:

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

1. Forest growth inputs

Creating the forest growth input object

We assume we have an appropriate forest object:

```
data(exampleforestMED)
```

a species parameter data frame:

```
data(SpParamsMED)
```

a soil input object:

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

and simulation control list:

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

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

```
x <- forest2growthInput(exampleforestMED, examplesoil, SpParamsMED, control)
```

1. Forest growth inputs

Structure of the growth input object (1)

The growth 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] "paramsGrowth"        "paramsAllometries"   "internalPhenology"   "internalWater"
## [17] "internalCarbon"      "internalAllocation"  "internalMortality"
```

1. Forest growth inputs

Structure of the growth input object (1)

The growth 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] "paramsGrowth"        "paramsAllometries"   "internalPhenology"   "internalWater"      
## [17] "internalCarbon"      "internalAllocation"  "internalMortality"
```

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*:

```
x$above
```

```
##          SP        N   DBH Cover   H        CR         SA   LAI_live LAI_expanded LAI_dead
## T1_148 148 168.0000 37.55    NA 800 0.6605196 437.032040 0.96734365   0.96734365        0
## T2_168 168 384.0000 14.60    NA 660 0.6055642  57.407064 0.86167321   0.86167321        0
## S1_165 165 749.4923    NA  3.75  80 0.8032817   1.251072 0.03928201   0.03928201        0
```

1. Forest growth inputs

Structure of the growth input object (2)

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

```
x$paramsGrowth
```

```
##           RERleaf RERsapwood  RERfineroot CCleaf CCsapwood CCfineroot RGRleafmax RGRsapwoodmax
## T1_148 0.01210607   4.93e-05 0.0009610199 1.5905      1.47        1.3       0.03            NA
## T2_168 0.01757808   4.93e-05 0.0072846640 1.4300      1.47        1.3       0.03            NA
## S1_165 0.02647746   4.93e-05 0.0072846640 1.5320      1.47        1.3       0.03         0.002
##        RGRcambiummax RGRfinerootmax    SRsapwood  SRfineroot      RSSG fHDmin fHDmax     WoodC
## T1_148   0.003688597            0.1 2.008417e-04 0.001897231 0.3725000     80    160 0.4979943
## T2_168   0.001680573            0.1 9.150612e-05 0.001897231 0.9500000     40    100 0.4740096
## S1_165            NA            0.1 1.350000e-04 0.001897231 0.7804035     NA     NA 0.4749178
##        MortalityBaselineRate
## T1_148                0.0050
## T2_168                0.0010
## S1_165                0.0015
```

1. Forest growth inputs

Structure of the growth input object (2)

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

```
x$paramsGrowth
```

```
##            RERleaf RERsapwood  RERfineroot CCleaf CCsapwood CCfineroot RGRleafmax RGRsapwoodmax
## T1_148 0.01210607   4.93e-05 0.0009610199 1.5905      1.47        1.3       0.03            NA
## T2_168 0.01757808   4.93e-05 0.0072846640 1.4300      1.47        1.3       0.03            NA
## S1_165 0.02647746   4.93e-05 0.0072846640 1.5320      1.47        1.3       0.03         0.002
##        RGRcambiummax RGRfinerootmax    SRsapwood  SRfineroot      RSSG fHDmin fHDmax     WoodC
## T1_148   0.003688597            0.1 2.008417e-04 0.001897231 0.3725000     80    160 0.4979943
## T2_168   0.001680573            0.1 9.150612e-05 0.001897231 0.9500000     40    100 0.4740096
## S1_165            NA            0.1 1.350000e-04 0.001897231 0.7804035     NA     NA 0.4749178
##        MortalityBaselineRate
## T1_148                0.0050
## T2_168                0.0010
## S1_165                0.0015
```

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`:

```
x$internalCarbon
```

```
##        sugarLeaf starchLeaf sugarSapwood starchSapwood
## T1_148 0.4029239 0.00925123    0.5738487      3.276375
## T2_168 0.3585751 0.00925123    1.0741383      3.280965
## S1_165 0.7223526 0.00925123    0.2857655      3.445161
```

2. Forest growth

Forest growth run

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

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

```
## Initial plant cohort biomass (g/m2): 7109.27
## Initial soil water content (mm): 291.257
## Initial snowpack content (mm): 0
## Performing daily simulations
##
##  Year 2001:....................................
##
## Final plant biomass (g/m2): 7424.35
## Change in plant biomass (g/m2): 315.086
## Plant biomass balance result (g/m2): 315.086
## Plant biomass balance components:
##   Structural balance (g/m2) 218 Labile balance (g/m2) 125
##   Plant individual balance (g/m2) 343 Mortality loss (g/m2) 28
## Final soil water content (mm): 273.035
## Final snowpack content (mm): 0
## Change in soil water content (mm): -18.2217
## Soil water balance result (mm): -18.2217
## 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) 95 Net rainfall (mm) 367
##   Infiltration (mm) 409 Runoff (mm) 9 Deep drainage (mm) 130
##   Soil evaporation (mm) 26 Transpiration (mm) 271
```

2. Forest growth

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:

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`, `Stand`, `Plants`	[same as `spwb` …]
`LabileCarbonBalance`	Components of the labile carbon balance
`PlantBiomassBalance`	Components of indvidual- and cohort-level biomass balance
`PlantStructure`	Structural variables (DBH, height, sapwood area…)
`GrowthMortality`	Growth and mortality rates
`subdaily`	Sub-daily outputs (not relevant here)

2. Forest growth

Plots and summaries

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

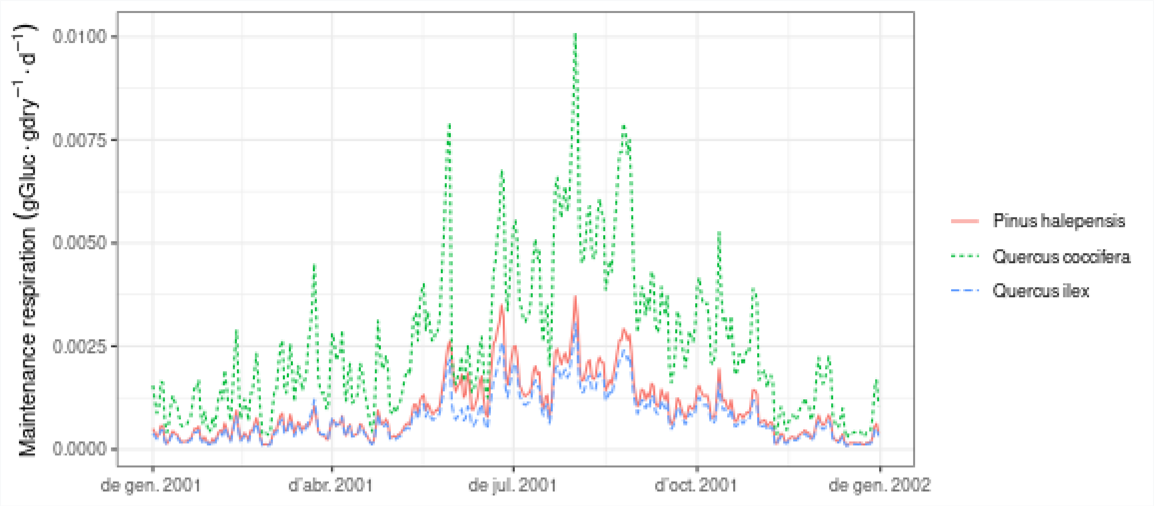
2. Forest growth

Plots and summaries

Users can inspect the output of `growth()` simulations using functions `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:

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



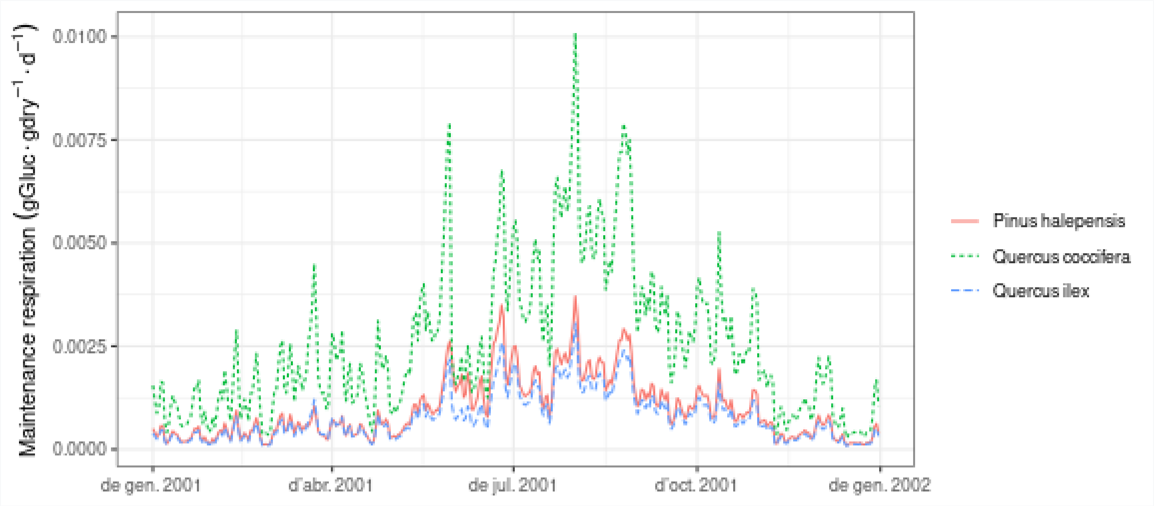
2. Forest growth

Plots and summaries

Users can inspect the output of `growth()` simulations using functions `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:

```
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.

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)
```

```
##                  SWC       ETR   E_T1_148   E_T2_168 FMC_T1_148 FMC_T2_168   BAI_T1_148 BAI_T2_168
## 2001-01-01 0.3009053 2.2054247 0.1711617 0.06695091   125.9066   93.07276 2.215983e-05          0
## 2001-01-02 0.3076413 2.5557696 0.3175461 0.18441164   125.8891   93.08217 1.080928e-09          0
## 2001-01-03 0.2989029 0.3980793 0.2382994 0.15495709   125.9559   93.04333 2.365990e-13          0
## 2001-01-04 0.2990861 2.1872094 0.2401646 0.06797015   125.9230   93.08681 3.576960e-11          0
## 2001-01-05 0.2983757 2.0845804 0.3581563 0.20087113   125.9423   93.07786 8.143438e-03          0
## 2001-01-06 0.3091887 2.6123669 0.2162016 0.18102197   125.8079   93.09805 1.964227e-03          0
##              DI_T1_148 DI_T2_168
## 2001-01-01 7.789547e-08         0
## 2001-01-02 1.184562e-11         0
## 2001-01-03 0.000000e+00         0
## 2001-01-04 4.580590e-13         0
## 2001-01-05 4.887803e-05         0
## 2001-01-06 2.176945e-05         0
```

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)
```

```
##                  SWC       ETR   E_T1_148   E_T2_168 FMC_T1_148 FMC_T2_168   BAI_T1_148 BAI_T2_168
## 2001-01-01 0.3009053 2.2054247 0.1711617 0.06695091   125.9066   93.07276 2.215983e-05          0
## 2001-01-02 0.3076413 2.5557696 0.3175461 0.18441164   125.8891   93.08217 1.080928e-09          0
## 2001-01-03 0.2989029 0.3980793 0.2382994 0.15495709   125.9559   93.04333 2.365990e-13          0
## 2001-01-04 0.2990861 2.1872094 0.2401646 0.06797015   125.9230   93.08681 3.576960e-11          0
## 2001-01-05 0.2983757 2.0845804 0.3581563 0.20087113   125.9423   93.07786 8.143438e-03          0
## 2001-01-06 0.3091887 2.6123669 0.2162016 0.18102197   125.8079   93.09805 1.964227e-03          0
##               DI_T1_148 DI_T2_168
## 2001-01-01 7.789547e-08         0
## 2001-01-02 1.184562e-11         0
## 2001-01-03 0.000000e+00         0
## 2001-01-04 4.580590e-13         0
## 2001-01-05 4.887803e-05         0
## 2001-01-06 2.176945e-05         0
```

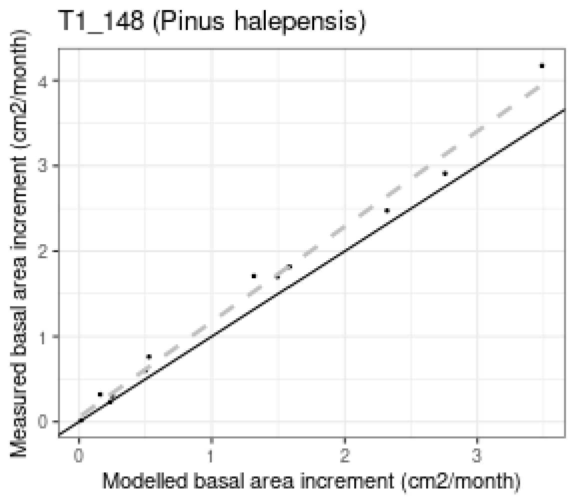
To specify observed growth data at *monthly* or *annual scale*, you should specify the first day of each month/year (e.g. `2001-01-01`, `2002-01-01`, etc for years) as row names in your observed data frame.

3. Evaluation of growth predictions

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")
```



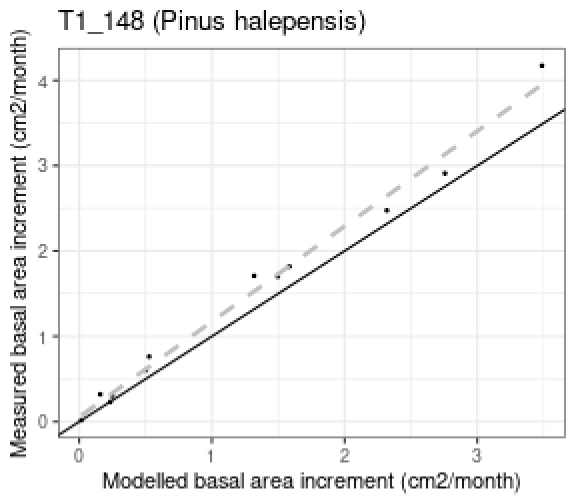
3. Evaluation of growth predictions

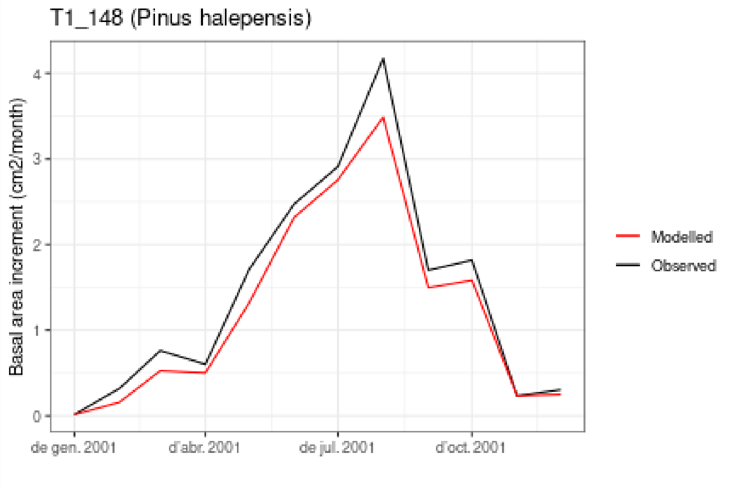
Evaluation plot

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")
```





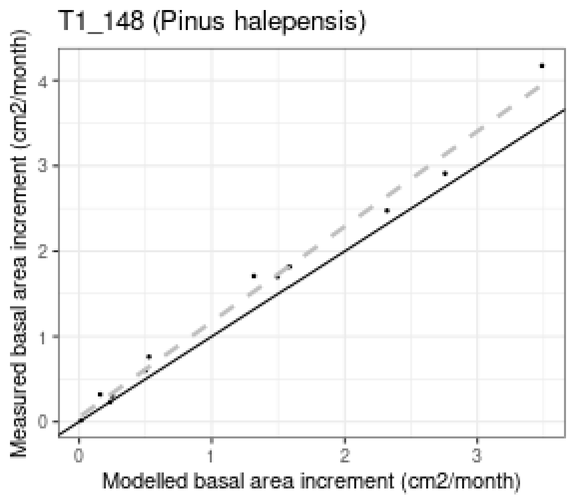
3. Evaluation of growth predictions

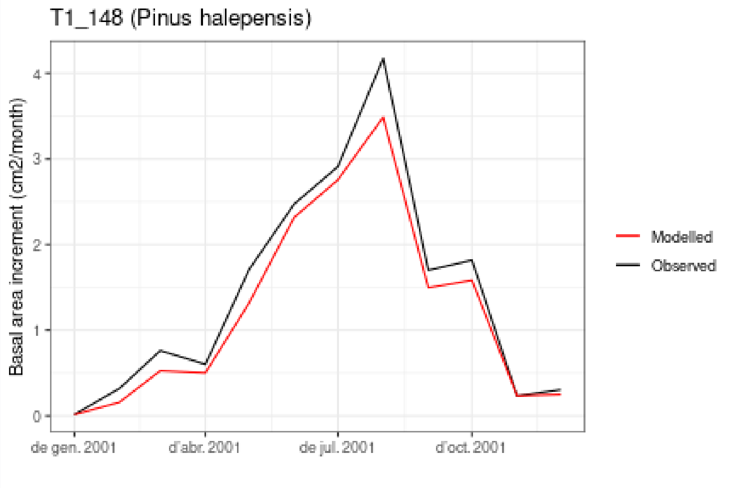
Evaluation plot

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.

3. Evaluation of growth predictions

Evaluation metrics

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         NSE     NSE.abs
##  12.0000000  -0.1977195 -13.9407205   0.1980247  13.9622393   0.9946258   0.9526141   0.8106137
```

4. Forest dynamics

Weather preparation

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

```
meteo = rbind(examplemeteo, examplemeteo, examplemeteo)
```

4. Forest dynamics

Weather preparation

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

```
meteo = rbind(examplemeteo, examplemeteo, examplemeteo)
```

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

```
row.names(meteo) = seq(as.Date("2001-01-01"),
                       as.Date("2003-12-31"), by="day")
```

4. Forest dynamics

Simulation

Remember: `fordyn()` operates on `forest` objects directly, instead of using an intermediary object (such as `spwbInput` and `growthInput`).

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

```
## Simulating year 2001 (1/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2002 (2/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2003 (3/3):  (a) Growth/mortality, (b) Recruitment
```

4. Forest dynamics

Simulation

Remember: `fordyn()` operates on `forest` objects directly, instead of using an intermediary object (such as `spwbInput` and `growthInput`).

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

```
## Simulating year 2001 (1/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2002 (2/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2003 (3/3):  (a) Growth/mortality, (b) Recruitment
```

Important: `fordyn()` calls function `growth()` internally for each simulated year.

4. Forest dynamics

Simulation

Remember: `fordyn()` operates on `forest` objects directly, instead of using an intermediary object (such as `spwbInput` and `growthInput`).

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

```
## Simulating year 2001 (1/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2002 (2/3):  (a) Growth/mortality, (b) Recruitment
## Simulating year 2003 (3/3):  (a) Growth/mortality, (b) Recruitment
```

Important: `fordyn()` calls function `growth()` internally for each simulated year.

The `verbose` option of the control parameters only affects function `fordyn()`, i.e. all console output from `growth()` is hidden.

4. Forest dynamics

Forest dynamics output

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()`.

4. Forest dynamics

Forest dynamics output

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

```
exampleforestMED
```

```
## $treeData
##   Species   N   DBH Height Z50  Z95
## 1     148 168 37.55    800 100  600
## 2     168 384 14.60    660 300 1000
##
## $shrubData
##   Species Cover Height Z50  Z95
## 1     165  3.75     80 200 1000
##
## $herbCover
## [1] 10
##
## $herbHeight
## [1] 20
##
## attr(,"class")
## [1] "forest" "list"
```

```
fd$NextForestObject
```

```
## $treeData
##   Species      DBH   Height        N Z50  Z95
## 1     148 38.29632 839.9312 165.4297 100  600
## 2     168 14.93838 671.0643 382.8201 300 1000
##
## $shrubData
##   Species Height    Cover Z50  Z95
## 1     165 76.549 3.356628 200 1000
##
## $herbCover
## [1] 10
##
## $herbHeight
## [1] 20
##
## attr(,"class")
## [1] "forest" "list"
```

4. Forest dynamics

Forest dynamics output

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

4. Forest dynamics

Forest dynamics output

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 TreeDensityLive
## 1    0              2              2               1               1        552.0000
## 2    1              2              2               1               1        550.7451
## 3    2              2              2               1               1        549.4950
## 4    3              2              2               1               1        548.2498
##   TreeBasalAreaLive DominantTreeHeight DominantTreeDiameter QuadraticMeanTreeDiameter
## 1          25.03330           800.0000             37.55000                  24.02949
## 2          25.27592           813.3815             37.79751                  24.17315
## 3          25.52068           826.7242             38.04689                  24.31752
## 4          25.76490           839.9312             38.29632                  24.46133
##   HartBeckingIndex ShrubCoverLive BasalAreaDead ShrubCoverDead BasalAreaCut ShrubCoverCut
## 1         53.20353       3.750000     0.0000000    0.000000000            0             0
## 2         52.38783       2.820976     0.1033233    0.004149262            0             0
## 3         51.60092       3.077311     0.1041971    0.004500137            0             0
## 4         50.84720       3.356628     0.1050688    0.004908933            0             0
```

4. Forest dynamics

Forest dynamics output

… and *species-level* statistics are shown using:

```
head(fd$SpeciesSummary)
```

```
##   Step Species              Name NumCohorts TreeDensityLive TreeBasalAreaLive ShrubCoverLive
## 1    0     148  Pinus halepensis          1        168.0000         18.604547             NA
## 2    0     165 Quercus coccifera          1              NA                NA       3.750000
## 3    0     168      Quercus ilex          1        384.0000          6.428755             NA
## 4    1     148  Pinus halepensis          1        167.1388         18.753984             NA
## 5    1     165 Quercus coccifera          1              NA                NA       2.820976
## 6    1     168      Quercus ilex          1        383.6063          6.521938             NA
##   BasalAreaDead ShrubCoverDead BasalAreaCut ShrubCoverCut
## 1   0.000000000             NA            0            NA
## 2            NA    0.000000000           NA             0
## 3   0.000000000             NA            0            NA
## 4   0.096629751             NA            0            NA
## 5            NA    0.004149262           NA             0
## 6   0.006693597             NA            0            NA
```

4. Forest dynamics

Forest dynamics output

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            Name        N      DBH   Height Z50  Z95
## 1    0   NA T1_148     148 Pinus halepensis 168.0000 37.55000 800.0000 100  600
## 2    0   NA T2_168     168    Quercus ilex 384.0000 14.60000 660.0000 300 1000
## 3    1 2001 T1_148     148 Pinus halepensis 167.1388 37.79751 813.3815 100  600
## 4    1 2001 T2_168     168    Quercus ilex 383.6063 14.71298 663.6731 300 1000
## 5    2 2002 T1_148     148 Pinus halepensis 166.2821 38.04689 826.7242 100  600
## 6    2 2002 T2_168     168    Quercus ilex 383.2130 14.82609 667.3719 300 1000
## 7    3 2003 T1_148     148 Pinus halepensis 165.4297 38.29632 839.9312 100  600
## 8    3 2003 T2_168     168    Quercus ilex 382.8201 14.93838 671.0643 300 1000
```

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

4. Forest dynamics

Forest dynamics output

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              Name        N      DBH   Height Z50  Z95
## 1    0   NA T1_148     148 Pinus halepensis 168.0000 37.55000 800.0000 100  600
## 2    0   NA T2_168     168     Quercus ilex 384.0000 14.60000 660.0000 300 1000
## 3    1 2001 T1_148     148 Pinus halepensis 167.1388 37.79751 813.3815 100  600
## 4    1 2001 T2_168     168     Quercus ilex 383.6063 14.71298 663.6731 300 1000
## 5    2 2002 T1_148     148 Pinus halepensis 166.2821 38.04689 826.7242 100  600
## 6    2 2002 T2_168     168     Quercus ilex 383.2130 14.82609 667.3719 300 1000
## 7    3 2003 T1_148     148 Pinus halepensis 165.4297 38.29632 839.9312 100  600
## 8    3 2003 T2_168     168     Quercus ilex 382.8201 14.93838 671.0643 300 1000
```

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

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

```
fd$DeadTreeTable
```

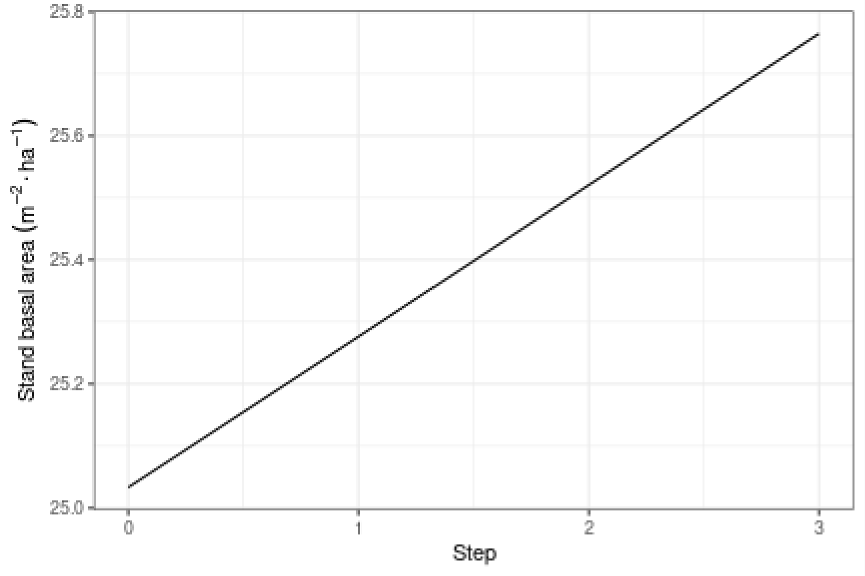
```
##   Step Year Cohort Species              Name         N      DBH   Height Z50  Z95
## 1    1 2001 T1_148     148 Pinus halepensis 0.8611814 37.79751 813.3815 100  600
## 2    1 2001 T2_168     168     Quercus ilex 0.3937029 14.71298 663.6731 300 1000
## 3    2 2002 T1_148     148 Pinus halepensis 0.8567669 38.04689 826.7242 100  600
## 4    2 2002 T2_168     168     Quercus ilex 0.3932992 14.82609 667.3719 300 1000
## 5    3 2003 T1_148     148 Pinus halepensis 0.8523751 38.29632 839.9312 100  600
## 6    3 2003 T2_168     168     Quercus ilex 0.3928960 14.93838 671.0643 300 1000
```

4. Forest dynamics

Summaries and plots

The provides a `plot` function for objects of class `fordyn`. For example, we can show the year-to-year variation in stand-level basal area using:

```
plot(fd, type = "StandBasalArea")
```

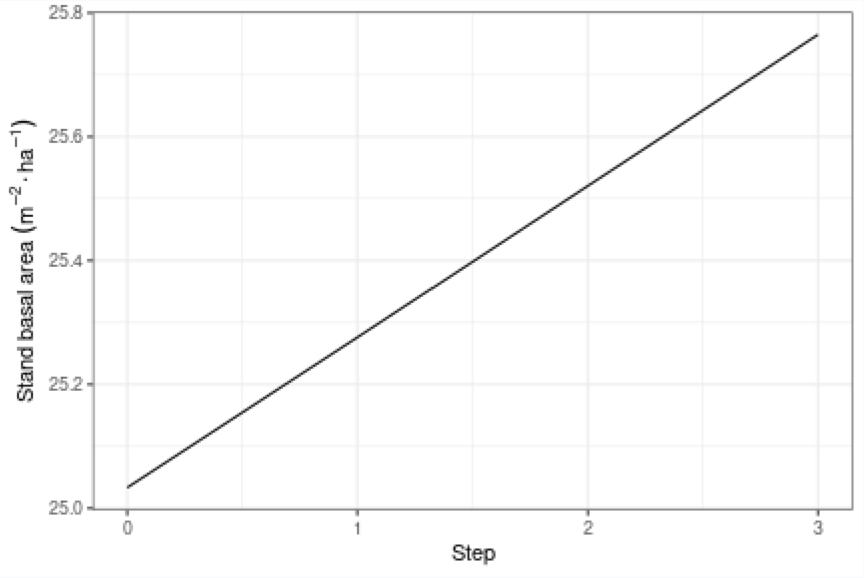


4. Forest dynamics

Summaries and plots

The provides a `plot` function for objects of class `fordyn`. For example, we can show the year-to-year variation in stand-level basal area using:

```
plot(fd, type = "StandBasalArea")
```



These plots are based on the *annual summaries* included in the output.

4. Forest dynamics

Summaries and plots

Remember: Function `fordyn()` makes internal calls to function `growth()` and stores the result in a vector called `GrowthResults`.

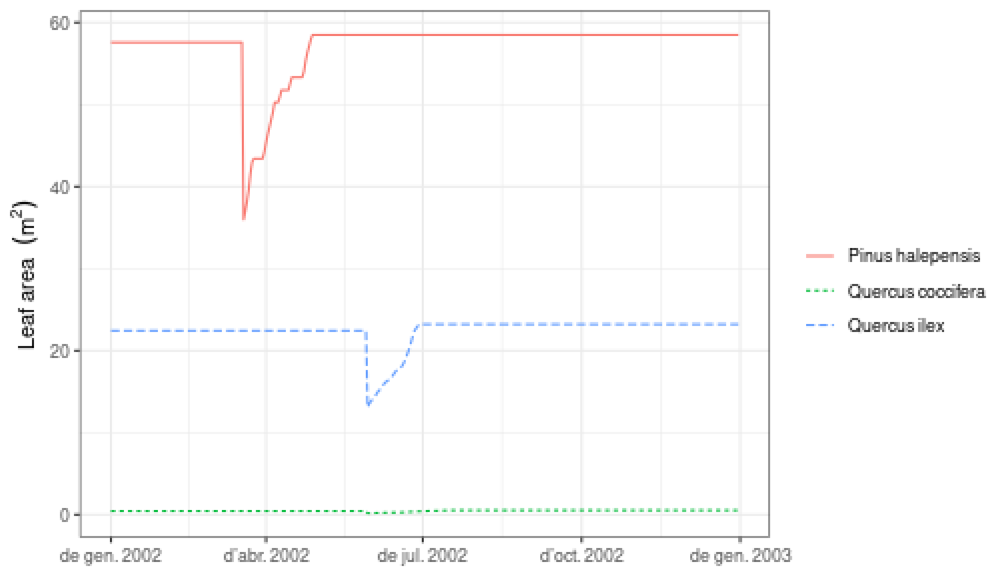
4. Forest dynamics

Summaries and plots

Remember: Function `fordyn()` makes internal calls to function `growth()` and stores the result in a vector called `GrowthResults`.

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

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

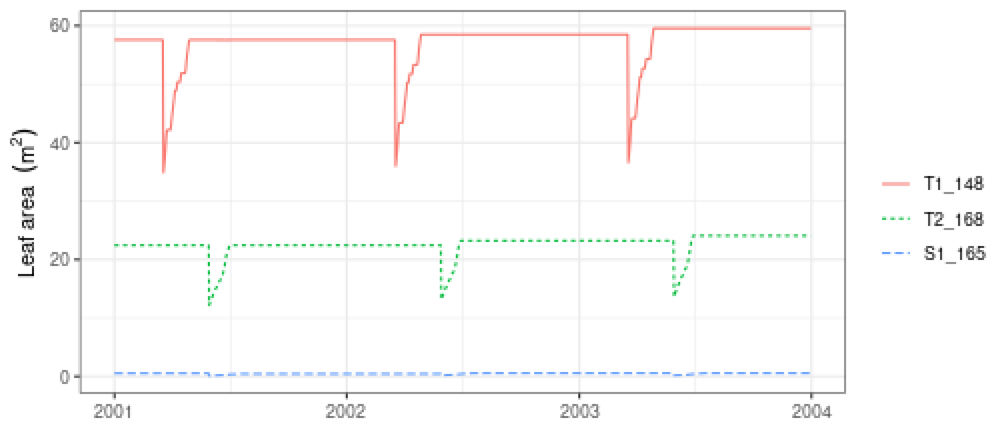


4. Forest dynamics

Summaries and plots

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

```
plot(fd, "LeafArea")
```

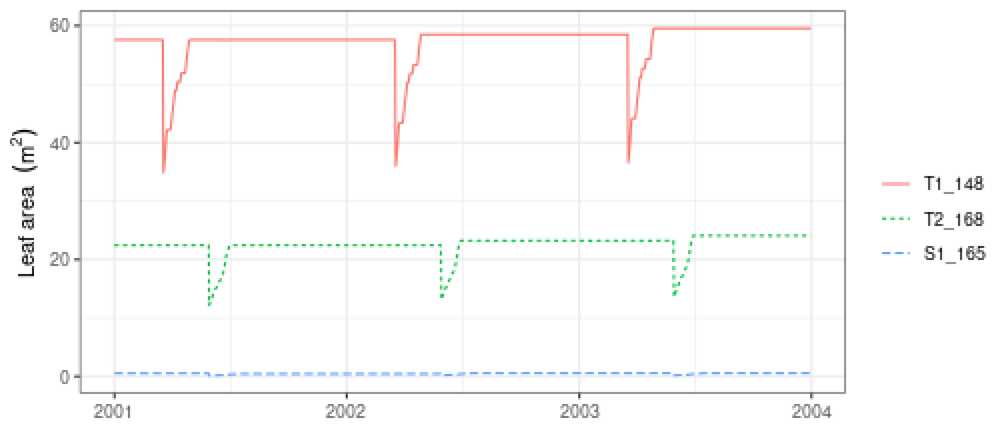


4. Forest dynamics

Summaries and plots

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

```
plot(fd, "LeafArea")
```



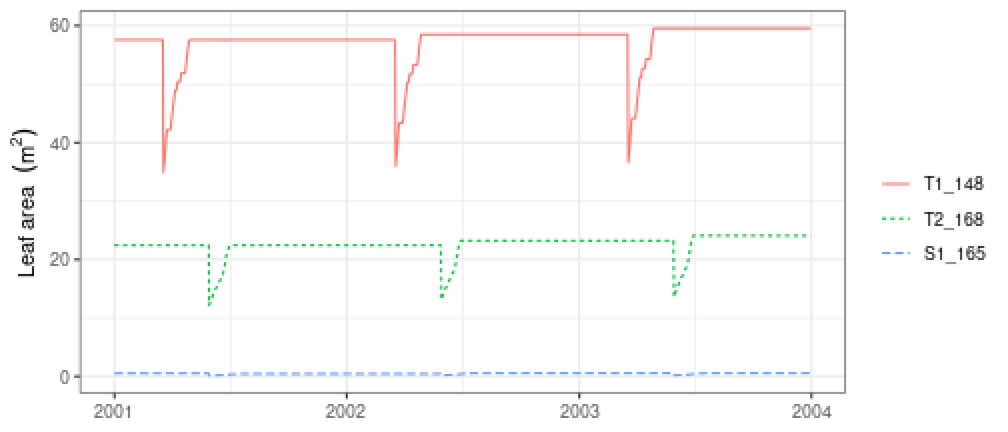
In this case, the `plot()` function assembles all the information from `GrowthResults` (accounting for cohort additions/deletions) and draws the plot.

4. Forest dynamics

Summaries and plots

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

```
plot(fd, "LeafArea")
```



In this case, the `plot()` function assembles all the information from `GrowthResults` (accounting for cohort additions/deletions) and draws the plot.

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

4. Forest dynamics

Forest dynamics including management

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

4. Forest dynamics

Forest dynamics including 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 with a flexible parameterization, a list with defaults provided by function `defaultManagementArguments()`.

4. Forest dynamics

Forest dynamics including 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 with a flexible parameterization, a list with defaults provided by function `defaultManagementArguments()`.

To run simulations with management we need to define (and modify) management arguments (see `?defaultManagementArguments`)…

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

4. Forest dynamics

Forest dynamics including 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 with a flexible parameterization, a list with defaults provided by function `defaultManagementArguments()`.

To run simulations with management we need to define (and modify) management arguments (see `?defaultManagementArguments`)…

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

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

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

4. Forest dynamics

Forest dynamics including 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 with a flexible parameterization, a list with defaults provided by function `defaultManagementArguments()`.

To run simulations with management we need to define (and modify) management arguments (see `?defaultManagementArguments`)…

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

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

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

When management is included, two additional tables are produced, e.g.:

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

4. Forest dynamics

Forest dynamics including management

Function `defaultManagementArguments()` returns a list with default values for *management parameters*:

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

4. Forest dynamics

Forest dynamics including management

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.

4. Forest dynamics

Forest dynamics including management

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.

Remenber: Besides using the in-built management function, you could program your own management function and specify its own set of parameters.

M.C. Escher - Up and down, 1947