Exercise 1 solution

Miquel De Cáceres

2022-06-13

Exercise setting

Objectives

1. Build `forest` objects from a tree data frame of forest inventory data
2. Retrieve soil physical properties from SoilGrids
3. Interpolate daily weather on the plot location

Data

Package **medfate** includes a data frame (`poblet_trees`), corresponding to forest inventory data in a dense holm oak forest.

- *Location*: Poblet (Catalonia, Spain); long/lat: 1.0219º, 41.3443º
- *Topography*: elevation = 850 m, slope = 15.1º, aspect = 15º
- *Plot*: Circular plot of 15-m radius
- *Tree data*: Stem diameter measurements on two plots: *control* and *managed*.

As a result of the abandonment of former coppicing in the area, there is a high density of stems per individual in the control plot.

The management involved a reduction of the number of stems per individual (*sucker cutback* or *selecció de tanys*).

Exercise solution

Step 1. Loading packages

We begin by loading packages **medfate**, **medfateland** and **meteoland** into the search path:

```
library(medfate)
library(medfateland)
library(meteoland)
```

Step 2. Load and inspect Poblet data

Normally, tree data would be in a **.csv** or **.xlsx** file. Here, we simply load the tree data from Poblet included in the package:

```
data("poblet_trees")
```

We can inspect its content, for example using:

```
summary(poblet_trees)
```

```
##   Plot.Code            Indv.Ref       Species           Diameter.cm
##  Length:717         Min.   :  1.0   Length:717         Min.   : 7.50
##  Class :character   1st Qu.: 45.0   Class :character   1st Qu.: 9.10
##  Mode  :character   Median : 97.0   Mode  :character   Median :11.10
##                     Mean   :103.4                      Mean   :11.62
##                     3rd Qu.:156.0                      3rd Qu.:13.40
##                     Max.   :261.0                      Max.   :26.00
```

The data frame includes tree data corresponding to three forest inventories:

```
table(poblet_trees$Plot.Code)
```

```
##
##     POBL_CTL POBL_THI_AFT POBL_THI_BEF
##          267          189          261
```

Step 3. Mapping trees from the control stand

We initialize an empty forest object using function `emptyforest()` from package **medfate**:

```
pobl_ctl <- emptyforest()
```

To fill data for element `treeData` in the `forest` object, we need to define a mapping from column names in `poblet_trees` to variables in `treeData`. The mapping can be defined using a **named string vector**, i.e. a vector where element names are variable names in `treeData` and vector elements are strings of the variable names in `poblet_trees`:

```
mapping <- c("Species.name" = "Species", "DBH" = "Diameter.cm")
```

Note: Using `"Species.name" = "Species"` we indicate that the function should interpret values in column `Species` as species names, not species codes.

We can now replace the empty `treeData` in `pobl_ctl` using functions `dplyr::filter()` and `forest_mapTreeTable()`:

```
pobl_ctl$treeData <- poblet_trees |>
  dplyr::filter(Plot.Code=="POBL_CTL") |>
  forest_mapTreeTable(mapping_x = mapping, SpParams = SpParamsMED)
```

Step 4. Check the mapping result

We can inspect the result using:

```
summary(pobl_ctl$treeData)
```

```
##    Species                N      Height             DBH          Z50
##  Length:267         Min.   :1   Mode:logical   Min.   : 7.50   Mode:logical
##  Class :character   1st Qu.:1   NA's:267      1st Qu.: 9.00   NA's:267
##  Mode  :character   Median :1                 Median :10.70
##                     Mean   :1                 Mean   :11.53
##                     3rd Qu.:1                 3rd Qu.:13.30
##                     Max.   :1                 Max.   :26.00
##    Z95
##  Mode:logical
##  NA's:267
##
##
##
##
```

One way to evaluate if the tree data is correctly specified is to display a summary of the `forest` object using the `summary()` function defined in **medfate** for this object class:

```
summary(pobl_ctl, SpParamsMED)
```

```
## Tree BA (m2/ha): 3.0179815  adult trees: 3.0179815  saplings: 0
## Density (ind/ha) adult trees: 267  saplings: 0  shrubs (estimated): 0
## Cover (%) adult trees: 42.1205627  saplings: 0  shrubs: 0  herbs: 0
## LAI (m2/m2) total: 0.544959  adult trees: 0.544959  saplings: 0  shrubs: 0  herbs: 0
## Fuel loading (kg/m2) total: 0.1421746  adult trees: 0.1421746  saplings: 0  shrubs: 0  herbs: 0
## PAR ground (%): NA  SWR ground (%): NA
```

Are the values of tree density, stand basal area and stand LAI acceptable for a dense oak forest?

Step 5. Specifying plot size

We were told that forest stand sampling was done using a *circular plot* whose radius was 15 m. We can calculate the sampled area using:

```
sampled_area <- pi*15^2
```

and use this information to map the tree data again, where we specify the parameter `plot_size_x`:

```
pobl_ctl$treeData <- poblet_trees |>
  dplyr::filter(Plot.Code=="POBL_CTL") |>
  forest_mapTreeTable(mapping_x = mapping, SpParams = SpParamsMED,
                      plot_size_x = sampled_area)
```

We run again the summary and check whether stand's basal area and LAI make more sense:

```
summary(pobl_ctl, SpParamsMED)
```

```
## Tree BA (m2/ha): 42.6957047  adult trees: 42.6957047  saplings: 0
## Density (ind/ha) adult trees: 3777.277316  saplings: 0  shrubs (estimated): 0
## Cover (%) adult trees: 100  saplings: 0  shrubs: 0  herbs: 0
## LAI (m2/m2) total: 5.6770407  adult trees: 5.6770407  saplings: 0  shrubs: 0  herbs: 0
## Fuel loading (kg/m2) total: 1.493419  adult trees: 1.493419  saplings: 0  shrubs: 0  herbs: 0
## PAR ground (%): NA  SWR ground (%): NA
```

Step 6. Adding tree heights

Another issue that we see is the `summary()` concerns percentage of PAR and SWR that reaches the ground, which have missing values. This indicates that light extinction cannot be calculated, in our case because tree heights are missing.

We should somehow estimate tree heights (in cm), for example using an allometric relationship:

```
poblet_trees$Height.cm <- 100 * 1.806*poblet_trees$Diameter.cm^0.518
summary(poblet_trees$Height.cm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   512.9   566.9   628.3   638.0   692.7   976.5
```

Once tree heights are defined, we can include them in our mapping vector:

```
mapping <- c("Species.name" = "Species", "DBH" = "Diameter.cm", "Height" = "Height.cm")
```

and rerun the tree data mapping.

Step 7. Mapping trees from the managed stand

Now we can address the *managed* stand, which has two codes corresponding to *before* and *after* the thinning intervention. Let us first deal with the pre-thinning state:

```
pobl_thi_bef <- emptyforest()
pobl_thi_bef$treeData <- poblet_trees |>
  dplyr::filter(Plot.Code=="POBL_THI_BEF") |>
  forest_mapTreeTable(mapping_x = mapping, SpParams = SpParamsMED,
                      plot_size_x = sampled_area)
```

```
## Warning in forest_mapTreeTable(dplyr::filter(poblet_trees, Plot.Code == : Taxon
## names that were not matched: Quercus humilis.
```

```
summary(pobl_thi_bef$treeData)
```

```
##    Species                N              Height           DBH
##  Length:261         Min.   :14.15   Min.   :512.9   Min.   : 7.50
##  Class :character   1st Qu.:14.15   1st Qu.:563.7   1st Qu.: 9.00
##  Mode  :character   Median :14.15   Median :628.3   Median :11.10
```

```
##                     Mean   :14.15   Mean   :635.5   Mean   :11.51
##                     3rd Qu.:14.15   3rd Qu.:681.9   3rd Qu.:13.00
##                     Max.   :14.15   Max.   :944.9   Max.   :24.40
##    Z50             Z95
##  Mode:logical   Mode:logical
##  NA's:261       NA's:261
##
##
##
##
```

Beware of the missing values in column `Species`

Step 8. Fixing species nomenclature

The `Species` variable contains two missing values. This will normally happen when some species cannot be identified. We can verify if this happens for other parts of the Poblet tree data:

```
sum(!(poblet_trees$Species %in% SpParamsMED$Name))
```

```
## [1] 4
```

If we display species counts we can identify which species is not being parsed:

```
table(poblet_trees$Species)
```

```
##
## Acer monspessulanum       Arbutus unedo Phillyrea latifolia     Quercus humilis
##                   2                 265                   6                   4
##        Quercus ilex
##                 440
```

In this case, the name used for the downy oak (*Quercus humilis*) is a synonym and needs to be replaced by its accepted name (*Quercus pubescens*), e.g.:

```
poblet_trees$Species[poblet_trees$Species=="Quercus humilis"] <- "Quercus pubescens"
```

Step 8. Fixing species nomenclature

Now we repeat our mapping and check the results:

```
pobl_thi_bef$treeData <- poblet_trees |>
  dplyr::filter(Plot.Code=="POBL_THI_BEF") |>
  forest_mapTreeTable(mapping_x = mapping, SpParams = SpParamsMED,
                      plot_size_x = sampled_area)
summary(pobl_thi_bef, SpParamsMED)
```

```
## Tree BA (m2/ha): 40.9224267  adult trees: 40.9224267  saplings: 0
## Density (ind/ha) adult trees: 3692.3946797  saplings: 0  shrubs (estimated): 0
## Cover (%) adult trees: 100  saplings: 0  shrubs: 0  herbs: 0
## LAI (m2/m2) total: 5.5833511  adult trees: 5.5833511  saplings: 0  shrubs: 0  herbs: 0
## Fuel loading (kg/m2) total: 1.4629714  adult trees: 1.4629714  saplings: 0  shrubs: 0  herbs: 0
## PAR ground (%): NA  SWR ground (%): NA
```

Like the control plot, the `summary()` indicates a dense oak forest.

Step 9. Mapping trees from the managed stand

We can finally map tree data for the forest plot *after* the thinning intervention:

```
pobl_thi_aft <- emptyforest()
pobl_thi_aft$treeData <- poblet_trees |>
  dplyr::filter(Plot.Code=="POBL_THI_AFT") |>
  forest_mapTreeTable(mapping_x = mapping, SpParams = SpParamsMED,
                      plot_size_x = sampled_area)
summary(pobl_thi_aft, SpParamsMED)
```

```
## Tree BA (m2/ha): 31.6162035  adult trees: 31.6162035  saplings: 0
## Density (ind/ha) adult trees: 2673.8030439  saplings: 0  shrubs (estimated): 0
## Cover (%) adult trees: 100  saplings: 0  shrubs: 0  herbs: 0
## LAI (m2/m2) total: 4.5328748  adult trees: 4.5328748  saplings: 0  shrubs: 0  herbs: 0
## Fuel loading (kg/m2) total: 1.1915321  adult trees: 1.1915321  saplings: 0  shrubs: 0  herbs: 0
## PAR ground (%): NA  SWR ground (%): NA
```

And check the reduction caused by the thinning.

Step 10. Checking the number of cohorts

So far we have considered that each tree record should correspond to a woody cohort. We can check the number of tree cohorts in each `forest` structure using:

```
nrow(pobl_ctl$treeData)
```

```
## [1] 267
```

```
nrow(pobl_thi_bef$treeData)
```

```
## [1] 261
```

```
nrow(pobl_thi_aft$treeData)
```

```
## [1] 189
```

This large amount of cohorts can slow done simulations considerably!

Step 11. Reducing the number of cohorts

One way of reducing the number of cohorts is via function `forest_mergeTrees()` from package **medfate**:

```
pobl_ctl <- forest_mergeTrees(pobl_ctl)
pobl_thi_bef <- forest_mergeTrees(pobl_thi_bef)
pobl_thi_aft <- forest_mergeTrees(pobl_thi_aft)
```

By default, the function will pool tree cohorts of the same species and diameter class (defined every 5 cm).

We can check the new number of tree cohorts using again:

```
nrow(pobl_ctl$treeData)
```

```
## [1] 9
```

```
nrow(pobl_thi_bef$treeData)
```

```
## [1] 11
```

```
nrow(pobl_thi_aft$treeData)
```

```
## [1] 8
```

Step 11. Reducing the number of cohorts

We can check whether stand properties were altered using the `summary()` function:

```
summary(pobl_thi_aft, SpParamsMED)
```

```
## Tree BA (m2/ha): 31.6162035  adult trees: 31.6162035  saplings: 0
## Density (ind/ha) adult trees: 2673.8030439  saplings: 0  shrubs (estimated): 0
## Cover (%) adult trees: 100  saplings: 0  shrubs: 0  herbs: 0
## LAI (m2/m2) total: 4.0969956  adult trees: 4.0969956  saplings: 0  shrubs: 0  herbs: 0
## Fuel loading (kg/m2) total: 1.0724731  adult trees: 1.0724731  saplings: 0  shrubs: 0  herbs: 0
## PAR ground (%): NA  SWR ground (%): NA
```

Function `forest_mergeTrees()` will preserve the stand density and basal area that the stand description had before merging cohorts. Other properties like leaf area index may be slightly modified.

Tip: It is advisable to reduce the number of woody cohorts before running simulation models in **medfate**.

Steps 12-13. Retrieving SoilGrids data

Retrieval of soil properties from SoilGrids can be done using function `add_soilgrids()` from package **medfateland**.

Assuming we know the plot coordinates, we first create an object `sf` (see package **sf**):

```
cc <- c(1.0219, 41.3443)
coords_sf <- sf::st_sf(geometry = sf::st_sfc(sf::st_point(cc), crs = 4326))
```

This object can be used to query SoilGrids using `add_soilgrids()`:

```
pobl_soil_props <- medfateland::add_soilgrids(coords_sf, widths = c(300, 700, 1000))
```

This function returns a data frame of soil properties:

```
pobl_soil_props
```

```
##   widths     clay     sand       om       bd  rfc
## 1    300 26.43333 31.06667 4.133333 1.166667 18.0
## 2    700 30.40000 29.75000 0.900000 1.440000 19.2
## 3   1000 31.60000 29.60000 0.610000 1.500000 20.9
```

Steps 14-15. Building the initialized soil object

This data frame is a physical description of the soil. Remember that the initialized soil data structure for **medfate** simulations is built using function `soil()`:

```
pobl_soil <- soil(pobl_soil_props)
```

We can inspect the soil definition using `summary()`.

```
summary(pobl_soil)
```

```
## Soil depth (mm): 2000
##
## Layer  1  [ 0  to  300 mm ]
##     clay (%): 26 silt (%): 38 sand (%): 31 organic matter (%): 4 [ Loam ]
##     Rock fragment content (%): 18 Macroporosity (%): 22
##     Theta WP (%): 18 Theta FC (%): 34 Theta SAT (%): 51 Theta current (%) 34
##     Vol. WP (mm): 44 Vol. FC (mm): 83 Vol. SAT (mm): 125 Vol. current (mm): 83
##     Temperature (Celsius): NA
##
## Layer  2  [ 300  to  1000 mm ]
##     clay (%): 30 silt (%): 39 sand (%): 30 organic matter (%): 1 [ Clay loam ]
##     Rock fragment content (%): 19 Macroporosity (%): 9
##     Theta WP (%): 19 Theta FC (%): 33 Theta SAT (%): 44 Theta current (%) 33
##     Vol. WP (mm): 106 Vol. FC (mm): 186 Vol. SAT (mm): 250 Vol. current (mm): 186
##     Temperature (Celsius): NA
##
## Layer  3  [ 1000  to  2000 mm ]
##     clay (%): 32 silt (%): 38 sand (%): 30 organic matter (%): 1 [ Clay loam ]
##     Rock fragment content (%): 21 Macroporosity (%): 6
##     Theta WP (%): 19 Theta FC (%): 33 Theta SAT (%): 44 Theta current (%) 33
##     Vol. WP (mm): 152 Vol. FC (mm): 263 Vol. SAT (mm): 347 Vol. current (mm): 263
##     Temperature (Celsius): NA
##
```

```
## Total soil saturated capacity (mm): 723
## Total soil water holding capacity (mm): 532
## Total soil extractable water (mm): 280
## Total soil current Volume (mm): 532
## Saturated water depth (mm): NA
```

SoilGrids usually underestimates the amount of rocks in the soil, because soil samples do not normally contain large stones or blocks. Realistic simulations should reduce the soil water holding capacity by increasing `rfc`. For example, here we will assume that the third layer contains 80% of rocks:

```
pobl_soil_props$rfc[3] = 80
```

If we rebuild the soil object and inspect its properties...

```
pobl_soil <- soil(pobl_soil_props)
summary(pobl_soil)
```

...we will see the decrease in water-holding capacity.

Steps 16-17. Interpolating weather

Obtaining daily weather data suitable for simulations is not straightforward either. Here we illustrate one way of obtaining such data with package **meteoland**.

We begin by building an object of class `sf` containing both the coordinates of our site as well as topographic variables.

```
pobl_sf <- coords_sf |>
  dplyr::mutate(elevation = 850, slope = 15.1, aspect = 15)
pobl_sf
```

```
## Simple feature collection with 1 feature and 3 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 1.0219 ymin: 41.3443 xmax: 1.0219 ymax: 41.3443
## Geodetic CRS:  WGS 84
##                  geometry elevation slope aspect
## 1 POINT (1.0219 41.3443)       850  15.1     15
```

The more difficult part of using package **meteoland** is to assemble weather data from surface weather stations into an object of class `stars`.

Here we will assume that such an object is already available, by using the example object provided in the **meteoland** package.

```
data("meteoland_interpolator_example")
```

Once we have an interpolator, obtaining interpolated weather for a set of target points is rather straightforward using function `interpolate_data()` from **meteoland**:

```
pobl_meteo <- pobl_sf |>
  meteoland::interpolate_data(meteoland_interpolator_example, verbose = FALSE)
```

The output of function `interpolate_data()` is also an `sf` object, with a new column `interpolated_data`:

```
pobl_meteo
```

```
## Simple feature collection with 1 feature and 4 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 1.0219 ymin: 41.3443 xmax: 1.0219 ymax: 41.3443
## Geodetic CRS:  WGS 84
## # A tibble: 1 x 5
##           geometry elevation slope aspect interpolated_data
##        <POINT [°]>     <dbl> <dbl>  <dbl> <list>
## 1 (1.0219 41.3443)       850  15.1     15 <tibble [30 x 13]>
```

We can access the weather data frame by subsetting the appropriate element of `interpolated_data`:

```
pobl_weather <- pobl_meteo$interpolated_data[[1]]
head(pobl_weather, 2)
```

```
## # A tibble: 2 x 13
##   dates                 DOY MeanTemperature MinTemperature MaxTemperature
##   <dttm>              <dbl>           <dbl>          <dbl>          <dbl>
## 1 2022-04-01 00:00:00    91            3.37          -2.21           6.99
## 2 2022-04-02 00:00:00    92            3.60          -4.01           8.54
## # i 8 more variables: Precipitation <dbl>, MeanRelativeHumidity <dbl>,
## #   MinRelativeHumidity <dbl>, MaxRelativeHumidity <dbl>, Radiation <dbl>,
## #   WindSpeed <dbl>, WindDirection <dbl>, PET <dbl>
```