Landscape- and regional-scale simulations (practice)

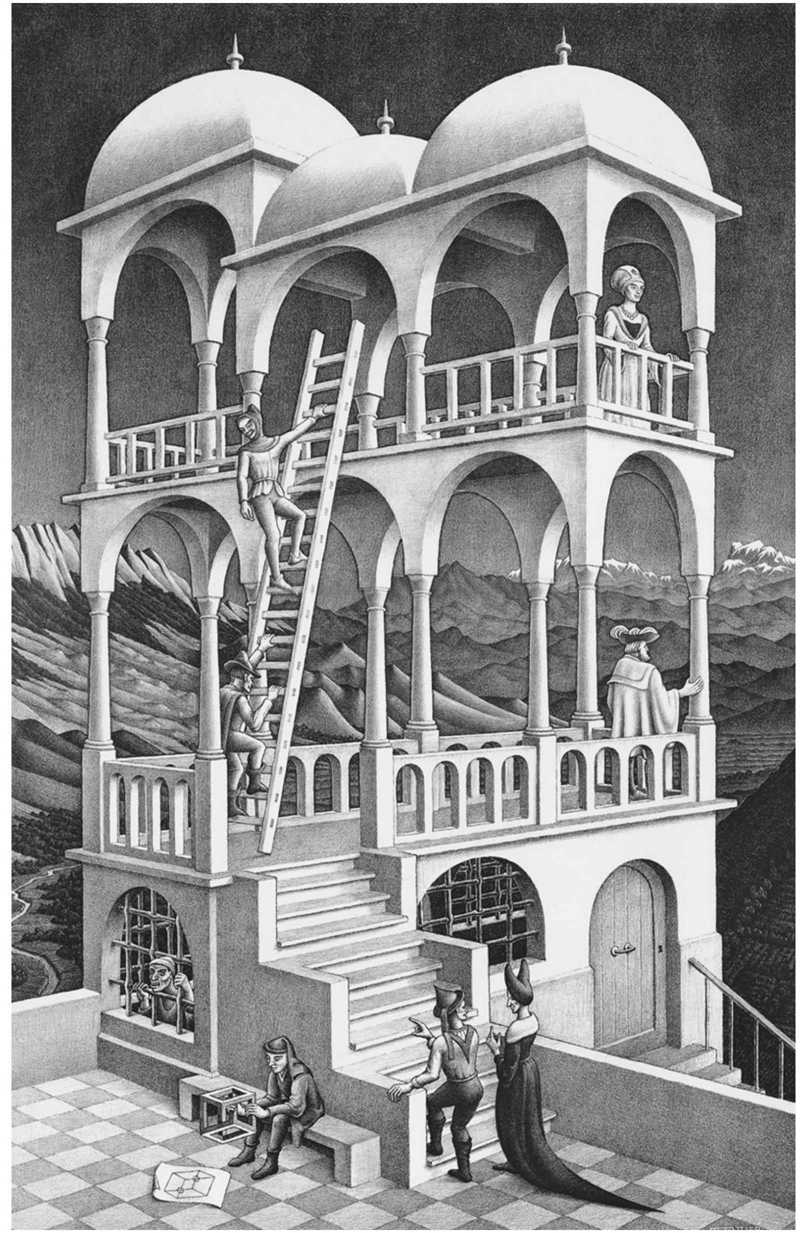
Miquel De Cáceres, Rodrigo Balaguer

Ecosystem Modelling Facility, CREAF

Outline

1. **Data structures in medfateland**
2. **Spatially-uncoupled simulations**
3. **Regional management scenarios**
4. **Watershed-level simulations**
5. **Creating spatial inputs I: forest inventory plots**
6. **Creating spatial inputs II: continuous landscapes**

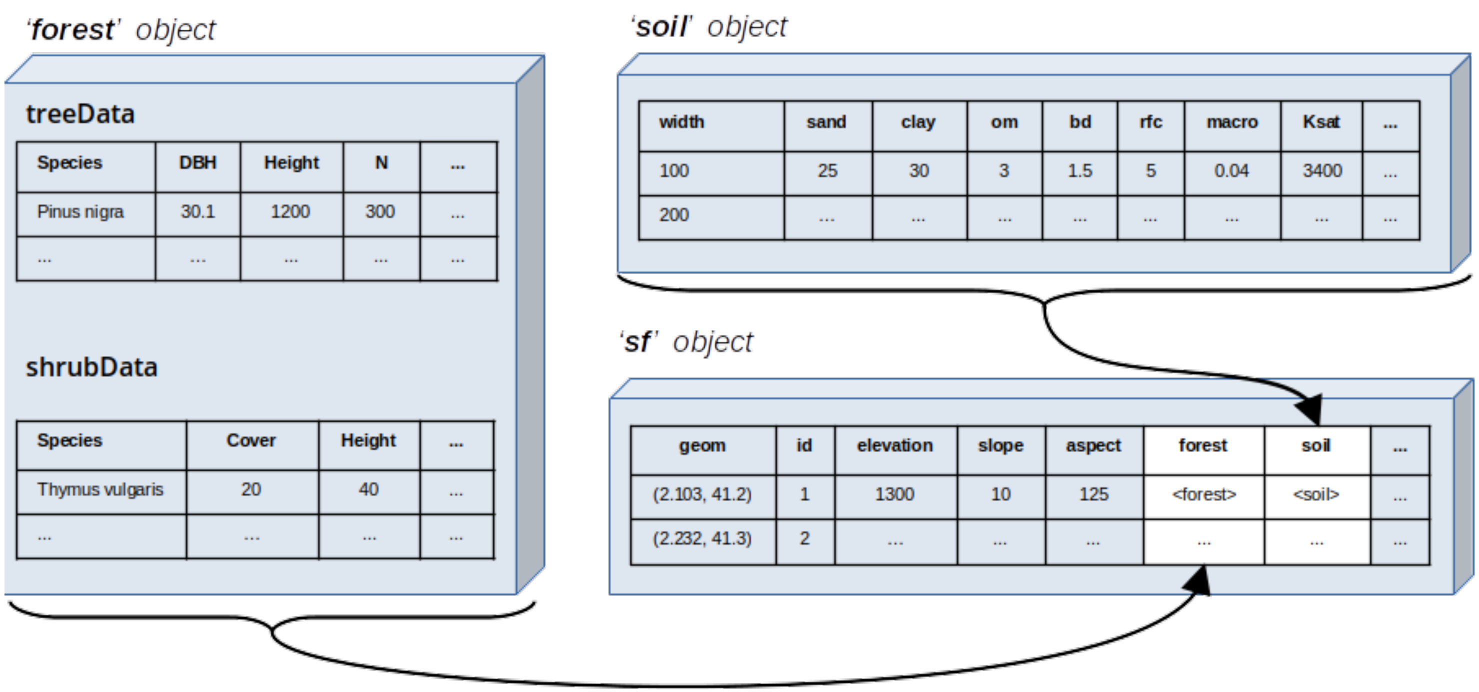
M.C. Escher - Belvedere, 1958



1. Data structures in medfateland

Spatial structures (1)

- Current versions of medfateland (ver. > 2.0.0) extensively use package **sf** (simple features) to represent spatial structures, where rows correspond to spatial units (normally point geometries) and columns include either *model inputs* (topography, forest, soil, weather forcing, etc.) or *model outputs*.
- Essentially, an `sf` object is a data frame with spatial (geometry) information and a coordinate reference system.
- Both `forest` and `soil` objects are nested in the corresponding columns of the `sf` object:



Spatial structures (2)

If we load the package we can inspect the structure of an example dataset with 100 forest inventory plots:

```
1  example_ifn
```

```
Simple feature collection with 100 features and 7 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 1.817095 ymin: 41.93301 xmax: 2.142956 ymax: 41.99881
Geodetic CRS:  WGS 84
# A tibble: 100 × 8
                  geom id        elevation slope aspect land_cover_type soil
 *         <POINT [°]> <chr>         <dbl> <dbl>  <dbl> <chr>           <list>
 1 (2.130641 41.99872) 081015_A1       680  7.73  281.  wildland        <df>
 2 (2.142714 41.99881) 081016_A1       736 15.6   212.  wildland        <df>
 3 (1.828998 41.98704) 081018_A1       532 17.6   291.  wildland        <df>
 4 (1.841068 41.98716) 081019_A1       581  4.79  174.  wildland        <df>
 5 (1.853138 41.98728) 081020_A1       613  4.76   36.9 wildland        <df>
 6 (1.901418 41.98775) 081021_A1       617 10.6   253.  wildland        <df>
 7 (1.937629 41.98809) 081022_A1       622 20.6   360   wildland        <df>
 8  (1.949699 41.9882) 081023_A1       687 14.4   324.  wildland        <df>
 9  (1.96177 41.98831) 081024_A1       597 11.8    16.3 wildland        <df>
10  (1.97384 41.98842) 081025_A1       577 14.6   348.  wildland        <df>
# ℹ 90 more rows
# ℹ 1 more variable: forest <list>
```

Accessing a given position of the `sf` object we can inspect `forest` or `soil` objects:

```
1  example_ifn$soil[[3]]
```

```
  widths     clay  sand   om       bd      rfc
1    300 25.76667 37.90 2.73 1.406667 23.84454
2    700 27.30000 36.35 0.98 1.535000 31.63389
3   1000 27.70000 36.00 0.64 1.560000 53.90746
4   2000 27.70000 36.00 0.64 1.560000 97.50000
```

Spatial structures (3)

To perform simulations on a gridded landscape we require both an `sf` object and an object `SpatRaster` from package **terra**, which defines the raster topology. For example, the following `sf` describes 65 cells in a small watershed:

```
example_watershed
```

```
Simple feature collection with 66 features and 14 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 401430 ymin: 4671870 xmax: 402830 ymax: 4672570
Projected CRS: WGS 84 / UTM zone 31N
# A tibble: 66 × 15
           geometry    id elevation slope aspect land_cover_type
 *      <POINT [m]> <int>     <dbl> <dbl>  <dbl> <chr>
 1 (402630 4672570)     1      1162 11.3   79.2  wildland
 2 (402330 4672470)     2      1214 12.4   98.7  agriculture
 3 (402430 4672470)     3      1197 10.4  102.   wildland
 4 (402530 4672470)     4      1180  8.12  83.3  wildland
 5 (402630 4672470)     5      1164 13.9   96.8  wildland
 6 (402730 4672470)     6      1146 11.2    8.47 agriculture
 7 (402830 4672470)     7      1153  9.26 356.   agriculture
 8 (402230 4672370)     8      1237 14.5   75.1  wildland
 9 (402330 4672370)     9      1213 13.2   78.7  wildland
10 (402430 4672370)    10      1198  8.56  75.6  agriculture
# ℹ 56 more rows
# ℹ 9 more variables: forest <list>, soil <list>, state <list>,
#   depth_to_bedrock <dbl>, bedrock_conductivity <dbl>, bedrock_porosity <dbl>,
#   snowpack <dbl>, aquifer <dbl>, crop_factor <dbl>
```

The following code defines a 100-m raster topology with the same CRS as the watershed:

```
r <-terra::rast(xmin = 401380, ymin = 4671820, xmax = 402880, ymax = 4672620,
                nrow = 8, ncol = 15, crs = "epsg:32631")
r
```

```
class       : SpatRaster
dimensions  : 8, 15, 1  (nrow, ncol, nlyr)
resolution  : 100, 100  (x, y)
extent      : 401380, 402880, 4671820, 4672620  (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 31N (EPSG:32631)
```

Weather forcing in medfateland

There are three ways of supplying weather forcing to simulation functions in **medfateland**, each with its own advantages/ disadvantages:

Supply method	Advantages	Disadvantages
A data frame as parameter `meteo`	Efficient both computationally and memory-wise	Assumes weather is spatially constant
A column `meteo` in `sf` objects	Allows a different weather forcing for each spatial unit	The resulting `sf` is often huge in memory requirements
An interpolator object of class `stars` (or a list of them) as issued from package **meteoland**	More efficient in terms of memory usage	Weather interpolation is performed during simulations, which entails some computational burden

Tip

- If a list of interpolator objects is supplied, each of the interpolators should correspond to a different, consecutive, non-overlapping time period (e.g. 5-year periods).
- Taken together, the interpolators should cover the simulated target period.
- The simulation function will use the correct interpolator for each target date.

2. Spatially-uncoupled simulations

Running spatially-uncoupled simulations

Since it builds on medfate, simulations using medfateland require species parameters and control parameters for local simulations:

```r
data("SpParamsMED")
local_control <- defaultControl()
```

We can specify the target simulation period as a vector of `Date` or subset the target plots:

```r
dates <- seq(as.Date("2001-01-01"), as.Date("2001-01-31"), by="day")
example_ifn_small <- example_ifn[1:5, ]
```

If we are interested in water (or energy) balance, we can use function `spwb_spatial()` as follows:

```r
res <- spwb_spatial(example_ifn_small, SpParamsMED, examplemeteo,
                    dates = dates, local_control = local_control)
```

The output is an `sf` object as well, where column `result` contains the results of calling `spwb()` and column `state` contains the final status of `spwbInput` objects:

```
Simple feature collection with 5 features and 3 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 1.828998 ymin: 41.98704 xmax: 2.142714 ymax: 41.99881
Geodetic CRS:  WGS 84
# A tibble: 5 × 4
             geometry id        state            result
          <POINT [°]> <chr>     <list>           <list>
1 (2.130641 41.99872) 081015_A1 <spwbInpt [19]> <spwb [10]>
2 (2.142714 41.99881) 081016_A1 <spwbInpt [19]> <spwb [10]>
3 (1.828998 41.98704) 081018_A1 <spwbInpt [19]> <spwb [10]>
4 (1.841068 41.98716) 081019_A1 <spwbInpt [19]> <spwb [10]>
5 (1.853138 41.98728) 081020_A1 <spwbInpt [19]> <spwb [10]>
```

Using summary functions (1)

Simulations with **medfate** can generate a lot of output. This can be reduced using `control` parameter, but simulation output with **medfateland** can require a lot of memory.

To save memory, it is possible to generate temporal summaries automatically after the simulation of each target forest stand, and avoid storing the full output of the simulation function (using `keep_results = FALSE`).

The key element here is the **summary function** (and possibly, its parameters), which needs to be defined and supplied.

In the following call to `spwb_spatial()` we provide the summary function for `spwb` objects available in **medfate**:

```
res_2 <- spwb_spatial(example_ifn_small, SpParamsMED, examplemeteo,
                      dates = dates, local_control = local_control,
                      keep_results = FALSE,
                      summary_function = summary.spwb, summary_arguments = list(freq="months"))
res_2
```

```
Simple feature collection with 5 features and 4 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 1.828998 ymin: 41.98704 xmax: 2.142714 ymax: 41.99881
Geodetic CRS:  WGS 84
# A tibble: 5 × 5
             geometry id        state            result summary
          <POINT [°]> <chr>     <list>           <list> <list>
1 (2.130641 41.99872) 081015_A1 <spwbInpt [19]> <NULL> <dbl [1 × 19]>
2 (2.142714 41.99881) 081016_A1 <spwbInpt [19]> <NULL> <dbl [1 × 19]>
3 (1.828998 41.98704) 081018_A1 <spwbInpt [19]> <NULL> <dbl [1 × 19]>
4 (1.841068 41.98716) 081019_A1 <spwbInpt [19]> <NULL> <dbl [1 × 19]>
5 (1.853138 41.98728) 081020_A1 <spwbInpt [19]> <NULL> <dbl [1 × 19]>
```

Using summary functions (2)

We can access the simulation summary for the first stand using:

```
res_2$summary[[1]]
```

```
                PET Precipitation     Rain     Snow  NetRain Snowmelt
2001-01-01 31.14173      74.74949 58.09884 16.65065 40.91681 13.09301
           Infiltration InfiltrationExcess SaturationExcess Runoff DeepDrainage
2001-01-01     54.00981                  0                0      0     32.61347
           CapillarityRise Evapotranspiration Interception SoilEvaporation
2001-01-01               0           30.34032     17.18203        5.405063
           HerbTranspiration PlantExtraction Transpiration
2001-01-01                 0        7.753223      7.753223
           HydraulicRedistribution
2001-01-01              0.01133329
```

Summaries can be generated *a posteriori* for a given simulation, using function `simulation_summary()`, e.g.:

```
simulation_summary(res, summary_function = summary.spwb, freq="months")
```

```
Simple feature collection with 5 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 1.828998 ymin: 41.98704 xmax: 2.142714 ymax: 41.99881
Geodetic CRS:  WGS 84
# A tibble: 5 × 3
             geometry id        summary
          <POINT [°]> <chr>     <list>
1 (2.130641 41.99872) 081015_A1 <dbl [1 × 19]>
2 (2.142714 41.99881) 081016_A1 <dbl [1 × 19]>
3 (1.828998 41.98704) 081018_A1 <dbl [1 × 19]>
4 (1.841068 41.98716) 081019_A1 <dbl [1 × 19]>
5 (1.853138 41.98728) 081020_A1 <dbl [1 × 19]>
```

Tip

Learning how to define summary functions is a good investment when using **medfateland**.

Continuing a previous simulation

The result of a simulation includes an element `state`, which stores the state of soil and stand variables at the end of the simulation. This information can be used to perform a new simulation from the point where the first one ended.

In order to do so, we need to update the state variables in spatial object with their values at the end of the simulation, using function `update_landscape()`:

```
example_ifn_mod <- update_landscape(example_ifn_small, res)
example_ifn_mod
```

```
Simple feature collection with 5 features and 8 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 1.828998 ymin: 41.98704 xmax: 2.142714 ymax: 41.99881
Geodetic CRS:  WGS 84
# A tibble: 5 × 9
                 geom id         elevation slope aspect land_cover_type soil
          <POINT [°]> <chr>          <dbl> <dbl>  <dbl> <chr>           <list>
1 (2.130641 41.99872) 081015_A1       680   7.73  281.  wildland        <soil>
2 (2.142714 41.99881) 081016_A1       736  15.6   212.  wildland        <soil>
3 (1.828998 41.98704) 081018_A1       532  17.6   291.  wildland        <soil>
4 (1.841068 41.98716) 081019_A1       581   4.79  174.  wildland        <soil>
5 (1.853138 41.98728) 081020_A1       613   4.76   36.9 wildland        <soil>
# ℹ 2 more variables: forest <list>, state <list>
```

Note that now the `sf` object contains a column `state` with initialized inputs.

Finally, we can call again the simulation function for a new consecutive time period:

```
dates <- seq(as.Date("2001-02-01"), as.Date("2001-02-28"), by="day")
res_3 <- spwb_spatial(example_ifn_mod, SpParamsMED, examplemeteo,
                    dates = dates, local_control = local_control)
```

Important

Function `update_landscape()` will also modify column `soil`.

M.C. Escher - Belvedere, 1958