Catflow: Tools for pre- and postprocessing for the hydrological model CATFLOW

Jan Wienhöfer
Karlsruhe Institute of Technology

Abstract

The **Catflow** R-package is a collection of utilities for the hydrological model CATFLOW. The objective of the package is to facilitate pre-processing operations when building up CATFLOW models, and post-processing of simulation runs. It collects tools that have been previously available as Matlab code, as well as newly added functions.

Keywords: utilities, preprocessing, postprocessing, CATFLOW, R.

Contents

Introduction

CATFLOW is a physically based, distributed model for simulating the dynamics of water and solutes in small rural catchments on the event and season time scale. The **Catflow** package is a collection of utilities for pre- and postprocessing functionality for CATFLOW, written for the R system for statistical computing (`http://www.R-project.org/`).

This vignette is intended to serve as a tutorial on how to use the tools of the **Catflow** package, providing a step-by-step example of a (simple) modelling session. For further information on individual functions, please refer to the respective help pages via `?NameOfTheFunction`.

The tutorial is organized in three parts:

- Part I explains how to create CATFLOW models for the simulation of one hillslope:
  - generate the model geometry, in terms of a system of curvilinear orthogonal coordinates, starting from a slope profile line (Section 1, p. 3),
  - write input files that relate to the modelled geometry or time-series which are to be included in the model (Section 2, p. 5),
- Part II details how to simulate three different types of macropores and prepare an appropriate model geometry (p. 12).
- Part III provides details on how to extract and visualise the results from CATFLOW simulations.

Please note: **Catflow** requires the packages **deSolve**, **RColorBrewer**, **splines**, **xts**, and **zoo**, which should be installed automatically together with **Catflow**.

Part I
Preprocessing for CATFLOW

This part describes how model geometry and other input files in the specific file formats for CATFLOW can be created.

To illustrate the usage of **Catflow**, we first load the package and create a sub-directory within the current working dir, where all the files for our sample CATFLOW session are collected. Here, we use a temporary folder and create a project directory `Catflow-TEST`). The next lines are R code that can be executed after copying to the console:

```
library("Catflow")
 # example project path, to a temporary folder
exampledir <- file.path(tempdir(),"Catflow-TEST")
 # create directory
if(!file_test("-d", exampledir)) dir.create(exampledir)
 # file path for input directory
indir <- file.path(exampledir, "in")
```

1. CATFLOW model geometry

To generate a CATFLOW model geometry, we need a slope profile line. This could either be obtained from a GIS (Geographic Information System), or it may be specified manually. Suppose we have the following slope line, represented by 20 points and associated values for slope elevation and width of the slope (Fig. 1):

```
# northing of slope line
  north <- seq(1,11, length=20)
# easting of slope line
  east  <- seq(2, 8, length=20)
# elevation of at points of slope line
  elev  <- approx(c(8,5),n=20)$y + sin((0:19)/2)/5
# width of slope at points of slope line (here: uniform)
  slopewidth <- rep(1,20)
```

Now suppose we want to generate a CATFLOW geometry from this slope line with a constant thickness of 2 m (`htyp = 1, dyy = 2`) and a discretization of 21 nodes to the lateral (`xsi`) and 11 nodes to the vertical (`eta`). We collect all the values needed, including the total area of the slope and the filename for the output, in a list:

```
simple.slope <- list( xh = north,
                      yh = east,
                      zh = elev,
                      bh = slopewidth,
```

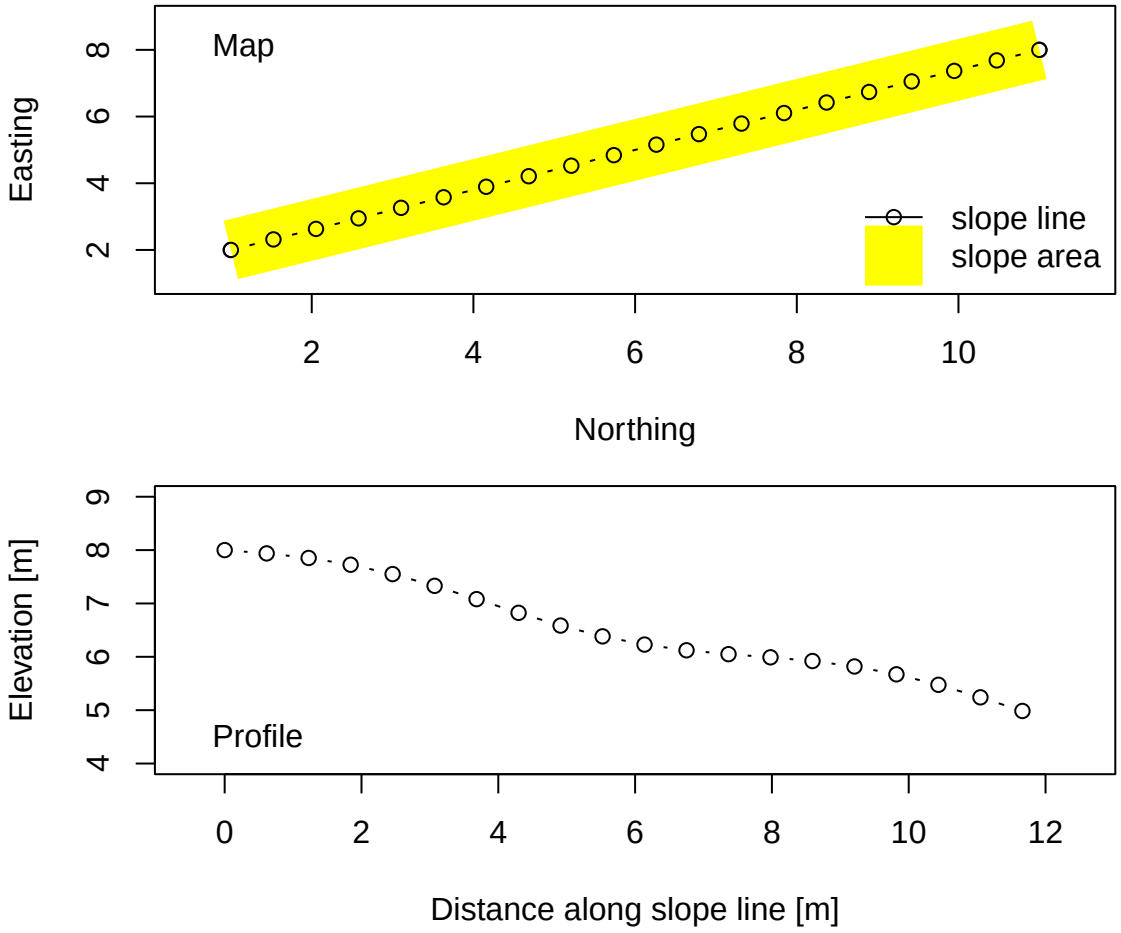


Figure 1: Schematic of slope line in map and profile view

```
                    tot.area = 12 ,
                    htyp = 1,
                    dyy = 2,
                    xsi = seq(0,1,length = 21),
                    eta = seq(0,1,length = 11),
                    out.file="test.geo"      )
```

With this list we are ready to generate a CATFLOW geometry using `make.geometry()`, which produces a text file for CATFLOW (`out.file="test.geo"`) in the specified directory (`project.path = indir`):

```
test.geom <- make.geometry(simple.slope, project.path = indir)
```

The following command generates a plot of the geometry (Fig. 2)

For further details, see `?make.geometry`. More elaborate examples of generating CATFLOW geometries are given in Part II in combination with the simulation of macropores.

```
plot.catf.geometry(test.geom, zooming = FALSE,
                   ylab="Elevation [m]",
                   xlab="Distance along slope line [m]")
```

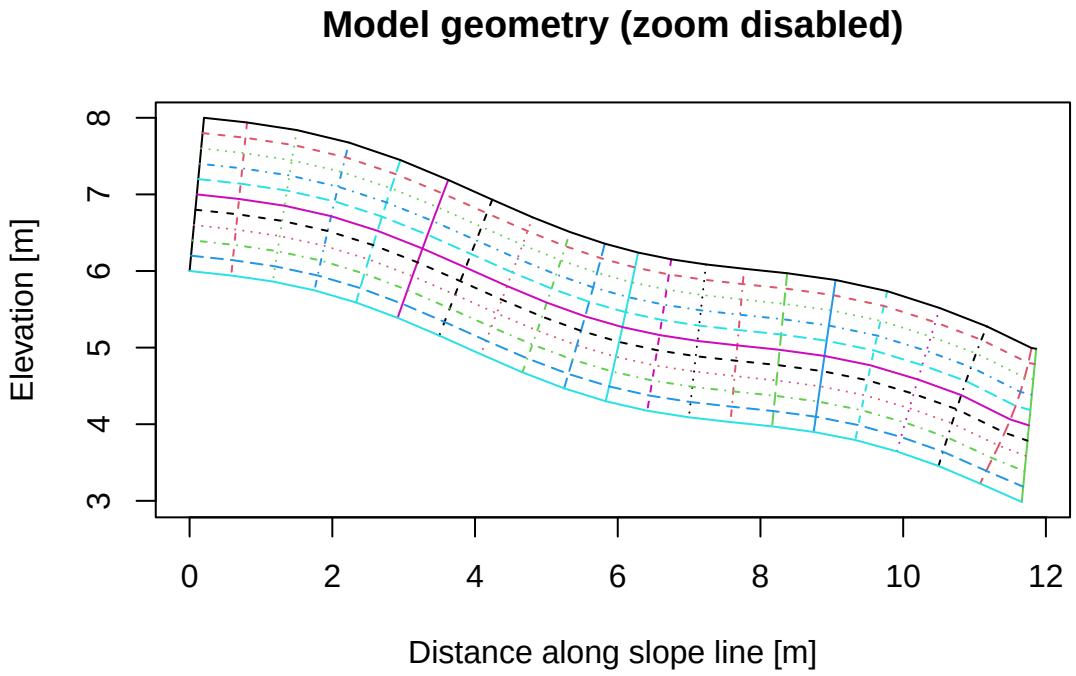


Figure 2: Model geometry generated with `make.geometry()` and plotted with `plot.catf.geometry()`.

2. Preparing CATFLOW input files

The **Catflow** package offers additional possibilities for writing input files besides the geometry file, namely:

- `write.facmat()`: multiplier matrices for K_s and θ_s, initial conditions, soil IDs;
- `write.precip()`: precipitation time series;
- `write.climate()`: climatic data time series;
- `write.printout()`: printout times;
- `write.surface.pob()`: surface node attributes;
- `write.control()`: project control files; and
- `write.CATFLOW.IN()`: the main input file.

To obtain a complete file structure for running CATFLOW, some more files are created using basic R commands (Section 2.7), but without using special functionality of the **Catflow** package.

2.1. Multiplier matrices for K_s and θ_s / Inital conditions

CATFLOW requires two files with multipliers, one for K_s and one for θ_s. These files can be generated using `write.facmat()`, which needs the discretization in terms of `eta` and `xsi`. These are contained in `test.geom` from Section 1, and are made available by attaching `test.geom`:

```
  attach(test.geom)    # attach the geometry to make 'eta' and 'xsi' available
  write.facmat(output.file=file.path(indir, "ksmult.dat"), eta, xsi)
  write.facmat(output.file=file.path(indir, "thsmult.dat"), eta, xsi)
```

It is also possible to specify non-uniform values – simply specify `fac` as a matrix of appropriate size. [1]

The same function may be used to specify initial conditions or soil type identifiers, but for these different header lines are needed:

```
# Initial conditions: Uniform Psi (0.8 m)
  write.facmat(output.file=file.path(indir, "soilhyd.ini"),
               eta, xsi,
               header=paste("PSI    ", 0,  1, length(eta), length(xsi), 1),
               fac = 0.8)
# Soil type IDs:
   write.facmat(output.file=file.path(indir, "soils.bod"),
               header= paste("BODEN",  length(eta), length(xsi), 1),
               eta, xsi,
               fac = matrix(c(rep(1, ceiling(length(eta)/2)),
                              rep(2,floor(length(eta)/2)) ),
               nrow = length(eta), ncol = length(xsi))  )
```

2.2. Precipitation time series

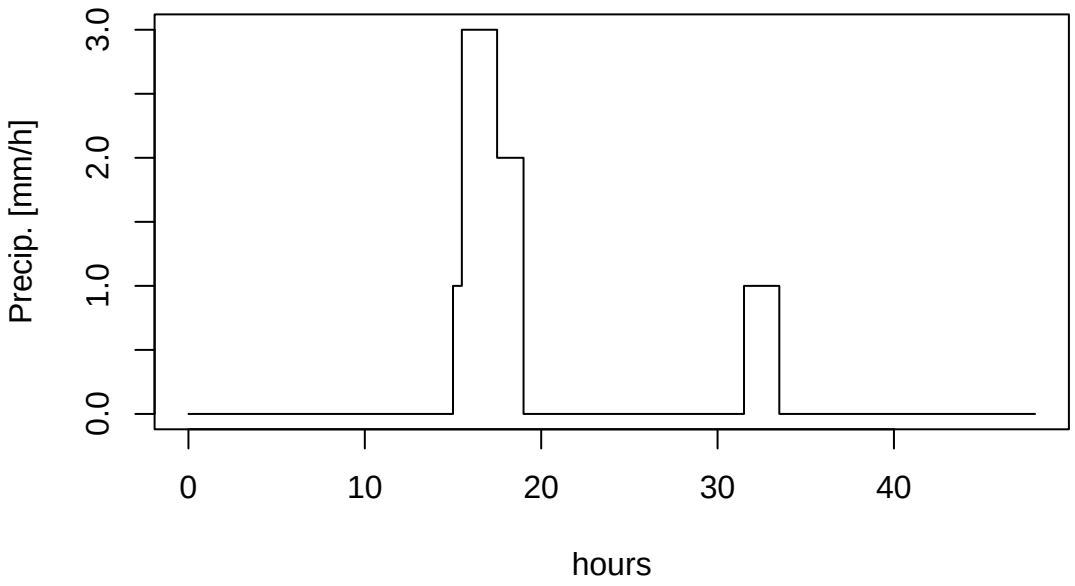
CATFLOW requires a precipitation record from which the rainfall intensities are interpolated between discrete time steps; rainfall intensities are thus only needed at those time steps when they are changing.

The function `write.precip()` converts a regular rainfall record, i.e. intensity at uniform time intervals, to the CATFLOW specific format:

```
# some artificial rainfall record
  raindat <- data.frame("hours" = seq(0,48, by=0.5),
                        "precip" = c(rep(0,30), 1, rep(3,4), rep(2,3),
```

[1] Note: CATFLOW currently cannot handle multipliers other than '1' for scaling saturated water content.

```
                                        rep(0,25), rep(1,4), rep(0,30)) )
plot(raindat, t="s", ylab="Precip. [mm/h]")
```




```
write.precip(raindat, file.path(indir, "TEST.rain.dat"),
             start.time= "01.01.2004 00:00:00" )
```

2.3. Climatic data time series

The function `write.climate()` writes a file with climatic data in the CATFLOW specific format. This mainly affects the header of the file, which holds some default parameters.

An example:

```
climadat <- data.frame(
     "hours" = seq(0,48, by=0.5),
     "GlobRad" =  ifelse(0 + 800 * sin((seq(0,48, by=0.5) - 8)*pi/12) > 0,
                         0 + 800 * sin((seq(0,48, by=0.5) - 8)*pi/12),  0),
     "NetRad" = NA ,
     "Temp" = 4 +  sin((seq(0,48, by=0.5) - 12)*pi/12)  ,
     "RelHum" = 70 + 10* sin((seq(0,48, by=0.5))*pi/12) ,
     "vWind"  =  rlnorm(97, 0,1) ,
     "dirWind" = runif(97, 0, 359)
     )
write.climate(climadat, file.path(indir, "TEST.clima.dat"),
              start.time= "01.01.2004 00:00:00" )
```

2.4. Printout times

To write a file with printout times in the CATFLOW specific format, `write.printout()` can be used. In the following example, we construct a printout time file for dumping all results (`flag = 1`) in intervals of 0.5 h

```
write.printout(output.file =  file.path(indir, "printout.prt"),
               start.time = "01.01.2004 00:00:00",
               end.time = "03.01.2004 00:00:00",
               intervall = 0.5, time.unit = "h",
               flag = 1)
```

2.5. Surface node attributes

The assignment of attributes (land-use, precipitation, climate, wind direction sectors) to the surface nodes of a CATFLOW model is done via the surface attribute file, which can be produced using `write.surface.pob()`.

In the following example, uniform values of land-use etc. are assigned to the surface nodes in `test.geom`, which should be still atttached, and so we can use the lateral discretization vector `xsi` directly:

```
  write.surface.pob(output.file =  file.path(indir, "surface.pob"),
                    xsi, lu = 33,
                    windid = rep(1,4))
```

2.6. Project control file and main control file

The function `write.control()` offers the possibility to create a project specific control file with the simulation control parameters and the input/output filenames. Additionally, a directory for the output is created.

In the following chunk we create a project control file with the input files that have been generated so far (simulation parameters can take default values here):

```
write.control(output.file = "TEST.example.in",
              project.path = exampledir,
              start.date = "01.01.2004 00:00:00.00",
              end.date = "03.01.2004 00:00:00",
              slope.in.list = list(
               slope1 = list( geo.file = "test.geo",
                              soil.file = "soils.bod",
                              ks.fac = "ksmult.dat",
                              ths.fac = "thsmult.dat",
                              macro.file = "profil.mak",
                              cv.file = "cont_vol.cv",
                              ini.file = "soilhyd.ini",
```

```
                                print.file = "printout.prt",
                                surf.file = "surface.pob",
                                bc.file = "boundary.rb")) )
```

Note that a output directory named 'out' has been added to the current working dir.

Finally, we need to write the main control file, which merely contains the name of the project control file:

```
write.CATFLOW.IN(control.files="TEST.example.in",
                 project.path = exampledir)
```

2.7. Other files

Some of the files given in the project control file have not been created so far, e.g. the hilllslope-specific in the `slope.in.list` of `write.control()`:

- macro.file: 'Catflow-TEST/in/profil.mak'
- cv.file: 'Catflow-TEST/in/cont_vol.cv'
- bc.file: 'Catflow-TEST/in/boundary.rb'

To complete the file structure, these are produced with the basic `cat` command (of course, you could as well use any text editor to create these files):

```
# macro.file = "profil.mak"
 cat(paste("1  0  2", "ari", "0.00 1.00 0.00 1.00 1  1.00 1.00 ", sep="\n"),
   file = file.path(indir, "profil.mak")  )
# cv.file = "cont_vol.cv"
 cat(paste("1", "0.8   0.9   0.98   1.0",sep="\n"),
   file = file.path(indir, "cont_vol.cv")  )
# bc.file = "boundary.rb"
 cat(paste("L", "1  0", "0. 1. 0",   " ",
           "R", "1  0", "0. 1. -10", " ",
           "T", "1  0", "0. 1. -99 "," ",
           "B", "1  0", "0. 1. 0",   " ",
           "S", "1  0", "0. 1. 0. 1. -99",   " ",
           "M", 0 , sep="\n"),
   file = file.path(indir, "boundary.rb")  )
```

The next bunch of files required by CATFLOW are the global simulation files defined in `global.in.list` for `write.control()`:

- winddir.file: 'Catflow-TEST/in/winddir.def'
- soildef.file: 'Catflow-TEST/in/soils.def'

- timeser.file: 'Catflow-TEST/in/timeser.def'
- lu.file: 'Catflow-TEST/in/landuse/lu_file.def'

These are created with the following chunks. First we generate the file for the definition of wind direction sectors:

```
  cat(paste("4",
            "240   0.81",
            " 50   0.78",
            " 80   0.97",
            "220   0.94", sep="\n"),
  file = file.path(indir, "winddir.def") )
```

This chunk writes a soil type definition for two soil types:

```
 cat(paste("2", "1 Loamy Sand, porosity 0.55, bulk dens 1 g/cm3",
           "1  800  1. 1.  1e-4   0.5 0.34 0.11  20. 0.70  0.050   1. 1. 1.",
           "4.05e-5  0.55 0.06 12.40 2.28 -6.00 8.00 1000.00 0.80",
           "0. 0. 0.", "0. 0. 0.", "0. 0. 0.",
           "2 Sandy Clay Loam (30% S, 40 % U; 30 % T)",
           "1  800  1. 1.  1e-4   0.5 0.34 0.11  20. 0.70  0.050   1. 1. 1.",
           "3.42e-6 0.48 0.08 0.96 1.5 -6.00 8.00 1200.00 0.80",
           "0. 0. 0.", "0. 0. 0.", "0. 0. 0.", sep="\n") ,
  file = file.path(indir, "soils.def") )
```

This chunk produces a a time-series definition file which links to the precipitation, land-use and climate records (see 2.2, 2.3):

```
 cat(paste("PREC", "1", "in/TEST.rain.dat", "",
           "BC", "0", "", "SINKS","0", "", "SOLUTE", "0", "",
           "LAND-USE", "in/landuse/lu_ts.dat", "",
           "CLIMATE", "1", "in/TEST.clima.dat", "", sep="\n"),
  file = file.path(indir, "timeser.def"))
```

Finally, we can prepare all the files related to land-use specifications, some of which are placed into their own sub-directory:

```
   if(!file_test("-d", file.path(indir, "landuse"))) {
    dir.create(file.path(indir, "landuse"))
    }
# pointer to land-use parameters
 cat(paste("3", "coniferous forest",
           "in/landuse/conif.par",
           sep = strrep(" ",13)),
   file = file.path(indir, "landuse", "lu_file.def") )
```

```
# time-series of land-use parameters
 cat(paste("01.01.2004 00:00:00.00", "in/landuse/lu_set1.dat",
           "01.01.2005 00:00:00.00", sep="\n"),
  file = file.path(indir, "landuse", "lu_ts.dat") )
# parameters of land-use type 'coniferous forest'
 cat(paste(
     paste("10", "KST", "MAK", "BFI", "BBG", "TWU", "PFH",
           "PALB", "RSTMIN", "WP_BFW", "F_BFW", sep= "    "),
     "0.    3.     1.    5.    0.95   5.0    5.0     0.15    1.    1.      1.",
     paste(c("1  ","366"),
     "2.     1.     1.     1.0   1.0    1.0     1.0   546.    0.05    30.",
     sep="    ", collapse="\n"),
    sep="\n"),
  file = file.path(indir, "landuse", "conif.par") )
# pointer to surface node attributes
 cat(paste(1, "33  3    %coniferous forest", sep = "\n"),
  file = file.path(indir, "landuse", "lu_set1.dat") )
```

Now that we have completed the preparation of input files we are ready to run CATFLOW on this simple example. First copy `CATFLOW.EXE` to the target directory, for from the **Catflow** package directory, and then run the program in the target directory. To do this within R, type:

```
file.copy(from = system.file("Catflow-TEST/CATFLOW.exe", package = "Catflow"),
          to = exampledir)
# changing into dir where CATFLOW should be executed
old.wd <- setwd(exampledir)
# run CATFLOW
system2(file.path(exampledir, "CATFLOW") )
# change back work dir
setwd(old.wd )
```

The simulation results are used in part III to demonstrate the post-processing facilities of the **Catflow** package.

Part II
Simulation of macropores as discrete flowpaths

The approach to simulate hydraulic effective structures on a given CATFLOW geometry comprises i) the generation of a simulation grid in the desired spatial resolution of the macroporous structures, ii) the actual simulation of macropores, iii) refinement of the discretization of simulation nodes and generation of a CATFLOW geometry file, and iv) representation of the simulated macropores as grids for defining "macroporous soil types" or multipliers for scaling hydraulic conductivities.

Three functions are available for the simulation of macroporous structures, and these are complemented with functions for creating a model discretization:

- `sim.mak()` and `discretize.mak()`: Vertical, tortuous structures
- `sim.pipe()` and `discretize.pipe()`: Horizontal, tortuous structures
- `sim.rectmak()` and `discretize.rect()`: Connected structures

3. Simple example for the simulation of macropores

The first step is to generate a simulation grid in fine resolution with the function `make.simgrid()`. Let us resume the example from above and produce a simulation grid for the same slope line (the column names in `test.slope` are for illustration purposes only):

```
test.slope <- data.frame( north = simple.slope$xh,
                          east = simple.slope$yh,
                          elev = simple.slope$zh,
                          slope.width = simple.slope$zh)

test.sim.grid <- make.simgrid(test.slope,
                              prof.depth = 2,
                              dx.max=0.1,
                              dz.max = 0.1)
```

The spatial resolution of the resulting grid is determined by the desired maximal resolution and the length of the profile; in this case the horizontal resolution is 0.0997 m instead of the desired 0.1 m.

Let us now simulate some vertical structures with `sim.mak()` and visualize the results with `plot.macros()` (Fig. 3):

```
set.seed(2011)

sim1 <- with(test.sim.grid,
             sim.mak(xnew,
                     znew,
                     width,
                     ksmean=2.5e-6,
                     kmacro=1.33e-5,
                     x.step=10)
             )

with(test.sim.grid,
     plot.macros(xnew, znew, sim1[[1]])
   )
```

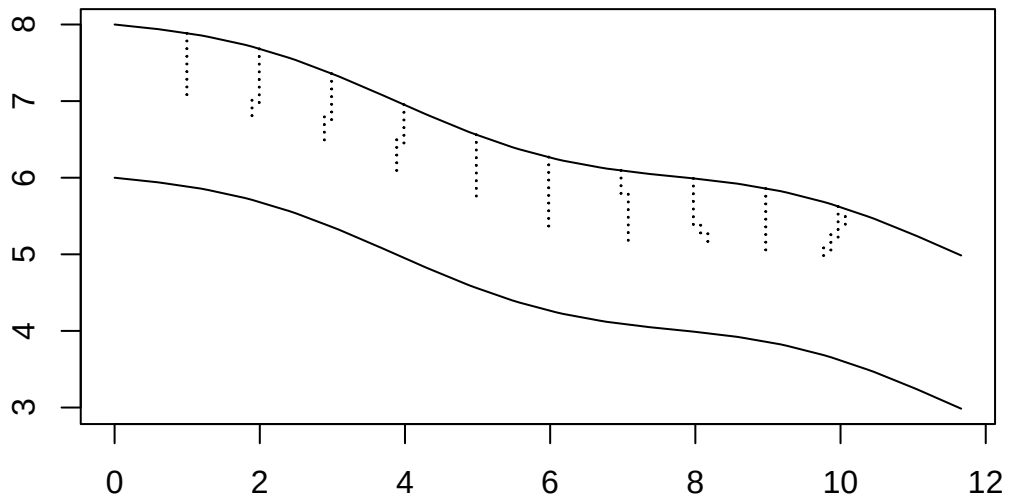


Figure 3: Schematic of slope line and simulated vertical macropores

Note that the grid `sim1` holds scaling factors of "1" for non-macroporous simulation nodes, while macropores have values above 1. Depending on the simulated number of macropores at a certain simulation node there can be different scaling factors for macropores. In the case of `sim1`, the scaling factors are distributed as follows:

```
table(round(as.vector(sim1[[1]]), 2))
```

Scaling factor	No. of nodes
1	2374
6.95	11
7.68	11
8.76	11
13.54	9
16.27	9
17.82	11
18.98	12
21.76	11
34.06	10
36.44	9

The simulated scaling factors may be directly used as scaling factors for saturated hydraulic conductivity after calculating a respective model discretization and adjusting the grid dimensions; or the simulated structures may be used to define different soil-types for matrix and macropores, as demonstrated further below.

The next step is to determine an appropriate model discretization to decrease the number of simulation nodes in the fine simulation grid. To keep the resolution fine around the macropores and make it more coarse in between, we apply the function `discretize.mak()` on the simulated macropore geometry:

Figure 4 shows the resulting discretization, which is fine vertically around the macropores, and fine at the top and around the endings of the macropores horizontally.

```
disc.sim1 <- with(test.sim.grid,
                  discretize.mak(
                   sim1,
                   maxdists = c(4,2),
                   plot=T)
                  )
```

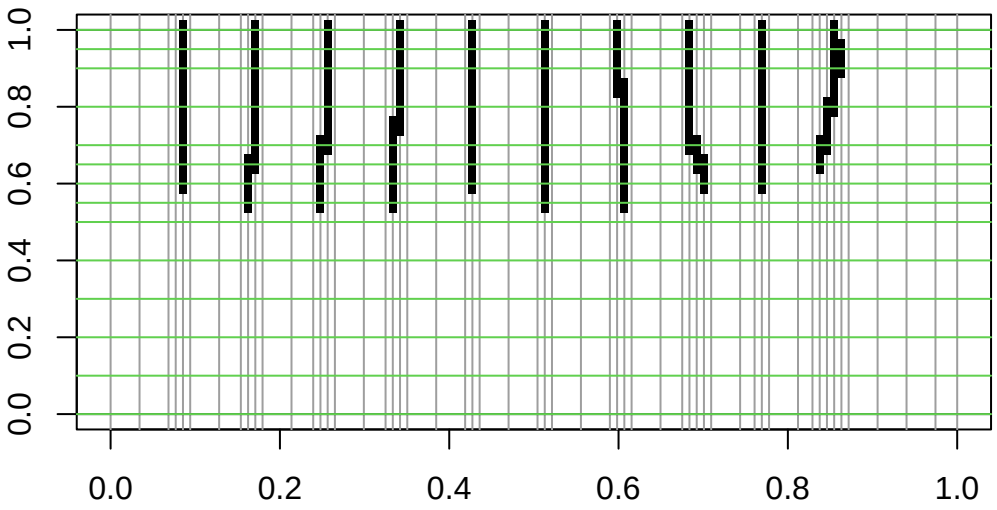


Figure 4: Schematic of discretization around simulated vertical macropores

With this discretization in terms of eta and xsi-vectors the corresponding geo-file for CATFLOW can be generated:

```
sim1.slope <- list(
      xh = test.slope$east,
      yh = test.slope$north,
      zh = test.slope$elev,
      bh  = test.slope$slope.width,
      tot.area = 12 ,
      numh = 1,
      htyp = 1,
      dyy = 2,
      xsi = disc.sim1[["xsi"]],
      eta = disc.sim1[["eta"]],
      out.file="sim1.geo"      )

sim1.geom <-  make.geometry(sim1.slope, project.path= indir,
                            out.file = "sim1.geo")
```

In order to complete the generation of input files for this model setup, we need to define the respective grids and soil type definitions for the simulated macropore geometry, which will differ in the dimensions of the grid for each simulated macropore structure. Towards this end we can make use of some of the functions that were already introduced in Section 2.

```
attach(sim1.slope)

# Multiplier grids (here all one)
write.facmat(output.file=file.path(indir,"thsmult_sim1.dat"), eta, xsi)
write.facmat(output.file=file.path(indir,"ksmult_sim1.dat"), eta, xsi)

# initial conditions: soilhyd.ini (Psi or Theta or earlier simulation)
write.facmat(output.file=file.path(indir,"soil_hyd_sim1.ini"),
             eta, xsi,
             header=paste("PSI   ", 0,  1, length(eta), length(xsi), numh),
             fac = 0.8)

# surface nodes (length xsi)
  write.surface.pob(output.file=file.path(indir,"surface_sim1.pob"),
                    xsi, lu=33)

detach(sim1.slope)
```

For the assignment of different soil types for matrix and macropores (or for scaling hydraulic conductivity), we need a grid with the dimension of eta and xsi that indicates the positions of macroporous nodes. This grid can be calculated from the simulated macropores in fine resolution and the new discretization using `mac.grid()`:

```
sim1.grid <- mac.grid(relfak = sim1[[1]],
                      xnew = test.sim.grid$xnew,
                       znew = test.sim.grid$znew,
                        xsi_new = disc.sim1[["xsi"]],
                         eta_new = disc.sim1[["eta"]],
                          plottin = FALSE)
```

In this case we want to assign different soil types for macropores and matrix. This is achieved by applying `assign.mac.soil()` on the `sim1.grid`. We will here define the soil matrix as soil type "1" and the macropores as a soil type "3":

```
# soil types (nodewise)
assign.mac.soil(sim1.grid,
                output.file=file.path(indir,"soilnodes_sim1.bod"),
                soil.macro = 3,
                thresh = 0.9)
```

Of course, we will have to add the definition of soil type "3" to the soil type definition file (repeating the first two entries from above):

```
cat(paste(
     "3", "1 Loamy Sand, porosity 0.55, bulk dens 1 g/cm3",
     "1  800  1. 1.  1e-4   0.5 0.34 0.11  20. 0.70  0.050   1. 1. 1.",
     "4.05e-5  0.55 0.06 12.40 2.28 -6.00 8.00 1000.00 0.80",
     "0. 0. 0.", "0. 0. 0.", "0. 0. 0.",
     "2 Sandy Clay Loam (30% S, 40 % U; 30 % T)",
     "1  800  1. 1.  1e-4   0.5 0.34 0.11  20. 0.70  0.050   1. 1. 1.",
     "3.42e-6 0.48 0.08 0.96 1.5 -6.00 8.00 1200.00 0.80",
     "0. 0. 0.", "0. 0. 0.", "0. 0. 0.",
     "3 Poremedium",
     "1 800 1.00 1.00 0.11 0.50 0.34 0.11 20.00 0.70 0.05 1. 1. 1.",
     "1.5e-4 0.4 0.057 11.4 2.28 -4.00 4.00 1600.00 0.80",
     "0. 0. 0.", "0. 0. 0.", "0. 0. 0.", sep="\n") ,
  file = file.path(indir,"soils.def") )
```

The other input files, e.g., for climate or land-use, can be left unchanged from the first example in Section 2, so only the main control file [2] and project control file with the respective output directory remain to be created:

```
# project control file,  default input.path="in"
write.control("TEST.sim1.in",
  output.path= "sim1out",
  project.path = exampledir,
  slope.in.list = list(
```

[2] Please note that this main control file overwrites the one created in Section 2!

```
    slope1 = list(
      geo.file= "sim1.geo"             ,  # slope geometry
      soil.file= "soilnodes_sim1.bod" ,  # soil type assignment
      ks.fac = "ksmult_sim1.dat"       ,  # multipliers for Ks
      ths.fac = "thsmult_sim1.dat"     ,  # multipliers for theta_s
      macro.file = "profil.mak"     ,     # macropore multipliers
      cv.file = "cont_vol.cv"       ,     # control volumes
      ini.file = "soil_hyd_sim1.ini",     # initial conditions (theta/psi)
      print.file = "printout.prt"   ,     # printout times
      surf.file = "surface_sim1.pob",     # surface attributes
      bc.file = "boundary.rb"             # boundary conditions
      )))

# main control file
write.CATFLOW.IN("TEST.sim1.in", project.path = exampledir)
```

Now CATFLOW can be run on this geometry with simulated macropores. The input files will be taken from the "Catflow-TEST/in" subdirectory, while the results will be collected in the "Catflow-TEST/sim1out" subdirectory.

3.1. Advanced example for simulation of macropores

As briefly indicated in the rather simple example above, the **Catflow** package offers various possibilites to produce model geometries with structural heterogeneity. In the following it will be described how the three approaches for the simulation of macroporous structures can be combined in order to build up more complex structures.

The idea is to subsequently apply the functions `sim.mak()`, `sim.pipe` or `sim.rect`; eventually with different function arguments. As an example, let us combine the simulation of vertical macropores from Section 3 with a simulation of horizontal macropores using `sim.pipe` on the simulation grid defined above, `test.sim.grid()`. Here we re not attaching the slope data again, but use `with()` to pass the slope data (xnew, znew) to the function:

```
set.seed(2011)
sim2 <- with(test.sim.grid,
             sim.pipe(
                relfak = sim1[[1]],
                xnew, znew,
                ml = 8,
                start.depth=1, p.up =0.1, p.down=0, x.step=5)
)
```

The resulting structures are plotted with `plot.macros()` (Fig. 5):

The grid `sim2` holds all of the simulated scaling factors and has the fine spatial resolution of the simulation grid.

The next step is to determine a appropriate model discretization with a reduced number of nodes. In the case of combined macropore simulations, the discretization functions are applied subsequently on each simulation result (Fig. 6), and the resulting eta- and xsi-vectors are combined. When doing this, care must be taken not to duplicate the regular nodes that are inserted at regular distances, which therefore are only introduced in the first discretization `disc.sim1` (same as above). The combined vectors need to be sorted and have duplicates removed. Note: Care should be taken to which digit the vectors are rounded - the idea is to remove eta / xsi-values that are so close to each other that they represent the same node. This closeness depends on how fine the discretization and how long the slope is, so the number of digits might need adaptation.

To visualize the combined discretization vectors, the following code chunk can be used:

From here on, the approach proceeds as described in Section 3. The next step would thus be the definition of a slope list with the slope line and the new model discretization:

```
sim2.slope <- list(
      xh = test.slope$east,
      yh = test.slope$north,
      zh = test.slope$elev,
      bh  = test.slope$slope.width,
      tot.area = 12 ,
      numh = 1,
      htyp = 1,
```

```
with(test.sim.grid,
      plot.macros(xnew, znew, sim2[[1]])
)
```

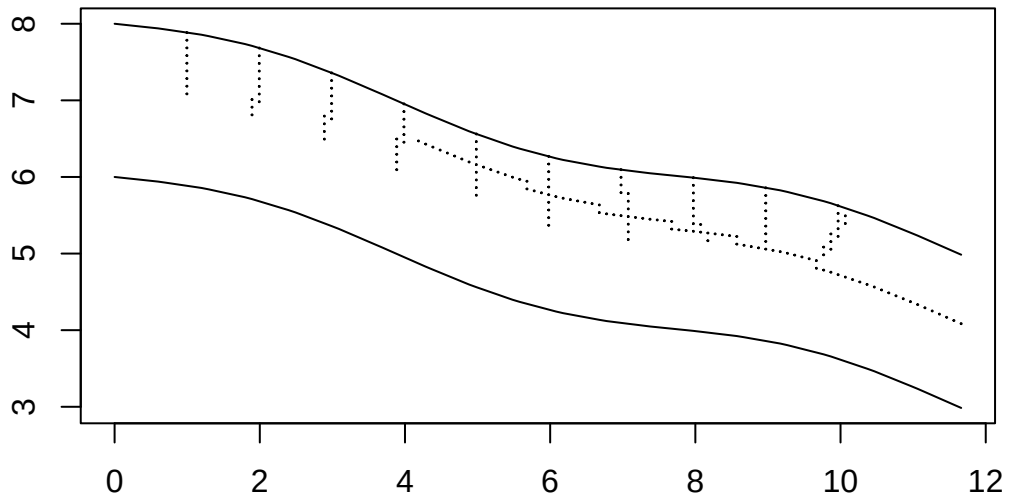


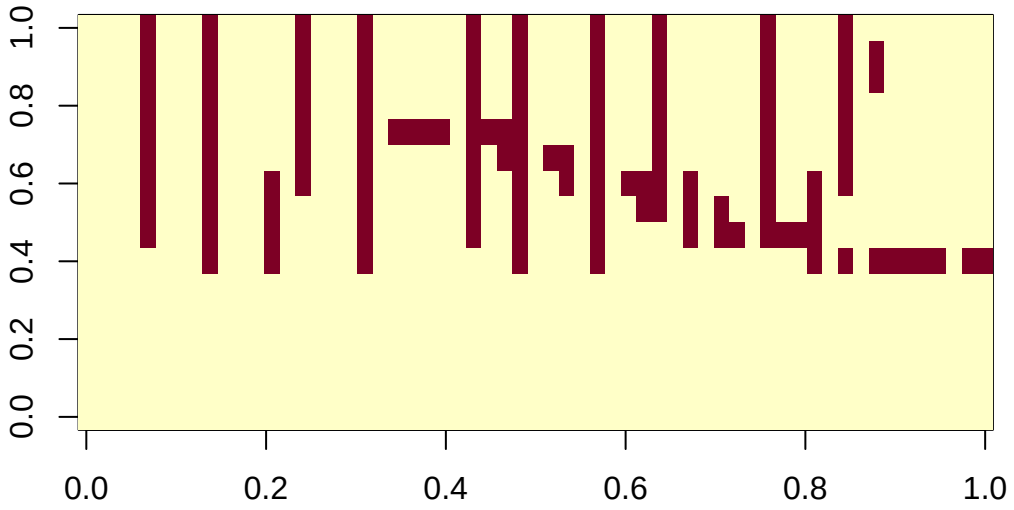
Figure 5: Schematic of slope line and combination of simulated vertical and horizontal macropores

```
        dyy = 2,
        xsi = disc.sim2[["xsi"]],
        eta = disc.sim2[["eta"]],
        out.file="sim2.geo"      )

  sim2.geom <-  make.geometry(sim2.slope, project.path= indir)
```

A grid that indicates the positions of macropres is generated with `mac.grid()`:

```
  sim2.grid <- mac.grid(relfak = sim2[[1]],
                        xnew = test.sim.grid$xnew,
                         znew = test.sim.grid$znew,
                          xsi_new = disc.sim2[["xsi"]],
                           eta_new = disc.sim2[["eta"]],
                            plottin = T)
```



⋮

Other files

The remaining steps would be the definition of control files, multiplier grids, surface attributes and the assignment of soil types in the same manner as described above.

Part III
Postprocessing of CATFLOW simulations

This part describes how the result files from CATFLOW simulations can be read into R for visualisation and further analyses.

4. Cleaning the simulation directory

By inspection of the output directory of the first example, we notice that some of the files produced by CATFLOW are not really useful:

```
print(dir(file.path(exampledir, "out")))

##  [1] "bilanz.csv" "c.out"      "evapo.out"  "fl_eta.out" "fl_xsi.out"
##  [6] "gang.out"   "hko.out"    "log.out"    "psi.fin"    "psi.out"
## [11] "qoben.out"  "relsat.out" "senken.out" "sko.out"    "theta.out"
## [16] "ve.out"     "vg_tab.out" "vx.out"
```

As we have not simulated solute transport or a drainage network, the corresponding output files are redundant, and we could therefore delete the files using the function `del.files`:

```
del.files(exampledir, file2del=c("ve.out", "vx.out", "c.out", "gang.out"))
```

Further cleaning of subdirectories will especially become useful with larger simulation projects, for example for simulations that differ only in some aspects (macropore geometry, land-use definitions, soil type parameters), but otherwise share a lot of input files. In these cases you might like to get rid of the input directory in the simulation folders, but retain the output files and some of the input. This can be achieved with the function `catf.batch.cleanup`:

```
catf.batch.cleanup(exampledir, indir = "./in", interact = TRUE)
```

5. Reading result files

Result files from CATFLOW simulations can be read using the functions :

- `read.catf.balance()`
- `read.surfrun.out()`
- `read.climate()`
- `read.catf.results()`
- `read.climate()`

- `read.evapo()`
- `read.precip()`
- `read.soil.mat()`
- `read.channelflow.out()`

6. Visualisation

Results from CATFLOW simulations can be visualised using the functions :

- `plot.catf.bal()`
- `plot.catf.grid()`
- `plot.catf.movie()`

7. Miscellaneous

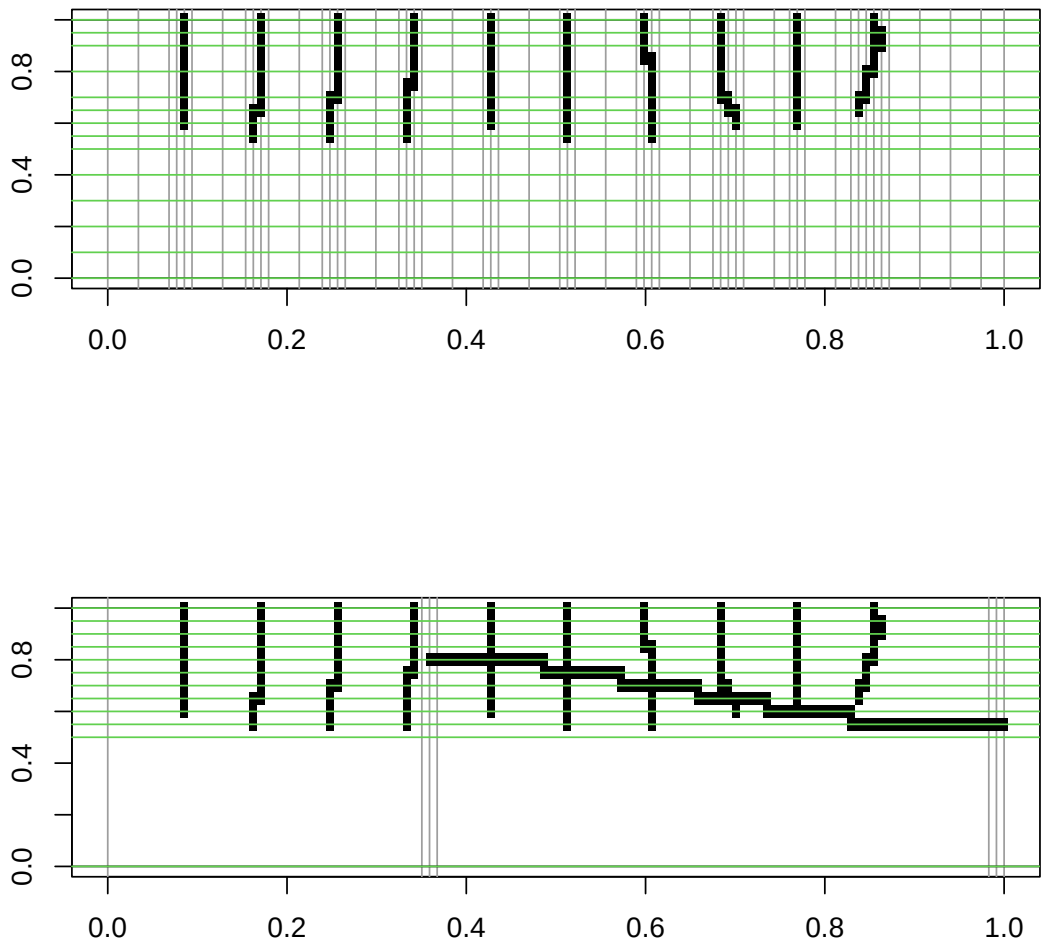
Further functions include:

- `use.psi.fin()`: Use final matric potential to assign initial conditions for a subsequent model run
- `read.facmat()`: Read a file with node-wise specification of values, e.g. initial conditions
- `get.realworld.coords()`: Calculate the real-world coordinates of a simulation domain
- `node2poly()`: Calculate polygons around computational nodes
- `color.codes()`: Color codes for plotting

The latter two functions are also used in the visualisation routines.

```
layout(1:2);

disc.sim1 <- discretize.mak(sim1, maxdists = c(4,2), plot=T)
disc.sim2 <- discretize.pipe(sim2, plot=T, reg=F)
```




```
disc.sim2[["xsi"]] <- sort(unique(round(c(disc.sim1[["xsi"]],
                                          disc.sim2[["xsi"]]), 3)))
disc.sim2[["eta"]] <- sort(unique(round(c(disc.sim1[["eta"]],
                                          disc.sim2[["eta"]]),2)) )
```

Figure 6: Individual model discretizations for `sim1` (top) and `sim2` (bottom).

```
## Plot discretization "by hand"
 plot(0:1,0:1,t="n", ann=F)
  abline(v=disc.sim2[["xsi"]], col=8)
  abline(h = c(0,1) )
  abline(h= disc.sim2[["eta"]], col=3)
  image(t(sim1[[1]]>1)[,nrow(sim1[[1]]):1], col=0:1, add=T)
  image(t(sim2[[1]]>1)[,nrow(sim2[[1]]):1], col=0:1, add=T)
```

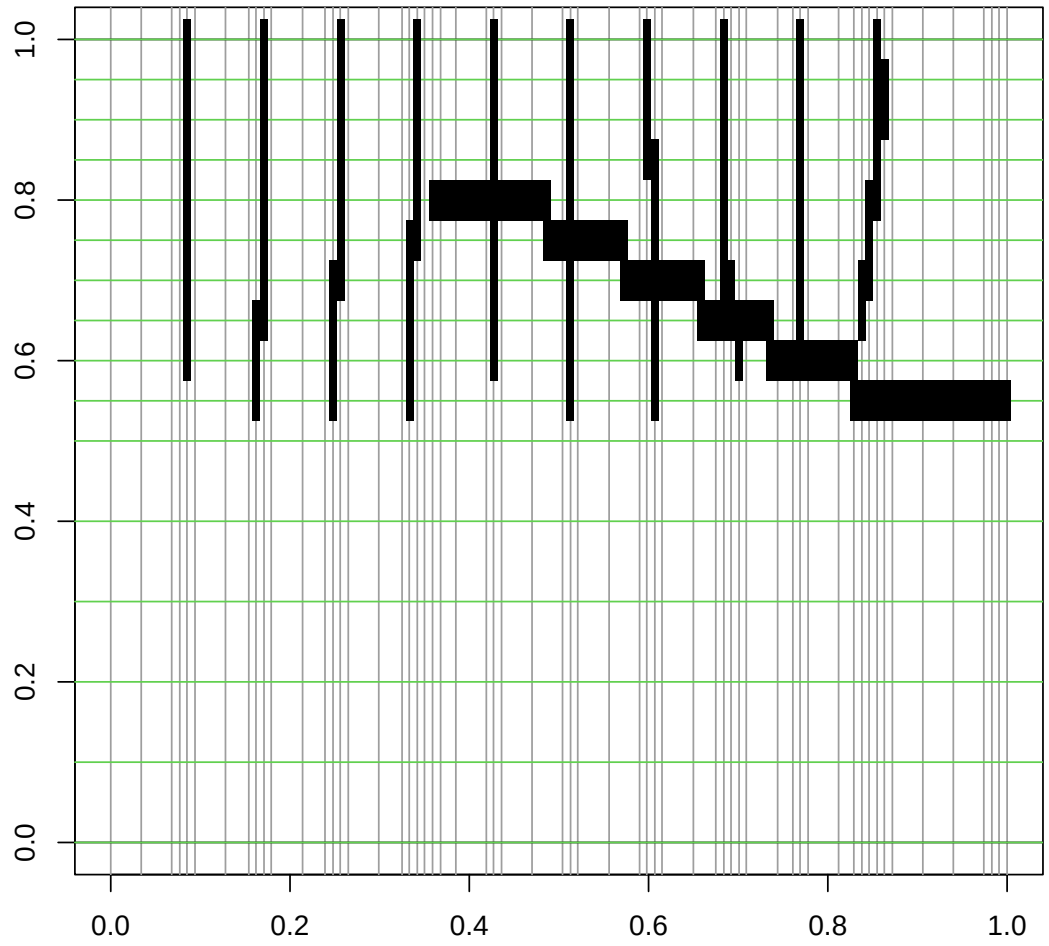


Figure 7: Combined model discretizations for `sim1` and `sim2`.

End of tutorial

A few things are left to tidy-up at the end of this tutorial session:

```
try(detach(test.geom), silent=TRUE)  # detach test geometry object
```

Computational details

The results in this vignette were obtained using the package **Catflow** 0.998, with R version 4.3.3 (2024-02-29 ucrt) and the packages **deSolve** 1.40, **RColorBrewer** 1.1–3, **splines** 4.3.3, **xts** 0.13.2 and **zoo** 1.8–12. R itself and the additional packages are available from CRAN at http://CRAN.R-project.org/.

Affiliation:

Jan Wienhöfer
Karlsruhe Institute of Technology
E-mail: jan.wienhoefer@kit.edu