[*this help is under construction*]

lass

landscape analysis simulation shell

www.ceam.es/lass

Introduction to lass language

See also the introductory paper by Pausas & Ramos (2005).

Lass is a simple command driven language for running vegetation models and analysing their results.

Lass has some similarities with the R/S language (although extremely simplified), making the life of R/S users a bit easier, and allowing compatibility.

Lass is case-sensitive.

Lass commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed, and the value is lost. An assignment also evaluates an expression and passes the value to a variable but the result is not automatically printed.

The assignment symbol is: <-

Variable names can be any string of letters and digits (a maximum of 25 characters), but a variable name cannot start with a digit, and cannot be any of the reserved words (which are: TRUE, FALSE, T, F). The strings can include the character "_" and but not the characters: ., $, /,-, .. nor those reserved for arithmetic operators.

The elementary *arithmetic operators* are the usual +, -, *, / and ^ (for raising to a power). In addition, some common arithmetic functions are available: log, exp, sqrt.

It is not always necessary to type the full name of the command: start typing the beginning of the name of the command and then press the tab key. This key will write the full name of the first command starting with the typed text; if this is not the required command, press the tab key again and it will go to the next command (in alphabetic order). Names of the variables can also be typed in a similar way. Optionally, by selecting "Online help" through the menu ("File/Online help"), you obtain the list of commands (in red) and variables (in blue) that can be chosen using the tab or the mouse.

Batch files (text files with lass commands in rows) can also be executed (see `source`). Commands are separated by a new line or by ";". Comments can be made by placing a hash mark (#); everything to the end of the line is a comment.

Character quantities and character vectors are denoted by a sequence of characters delimited by the double-quote character, e.g., "reprouter", "high", "yes".

A quick and brief help for any command can be accessed by: "commandname(?)" or help(commandname), e.g. mean(?) or help(mean). For a more detailed help see the help file in pdf form (lasshelp.pdf). A brief but useful online help can be accessed from the menu: "File/Online help".

Lass is designed as a command line interface (CLI). Furthermore, some commands (especially some related to simulation models) have an alternative graphic user interface (GUI). E.g., see species() and species.edit(), and run(... plot=T) in the library(melca).

Data structure. Lass has two main types of objects for storing data:

- Matrix s.l.: which can be: scalars (dim= 1 x 1), vectors (n x 1) or matrix s.s. (n x m). All values of a matrix are of the same type (numbers or strings). Matrix may have column names (see `setames`), but matrix row names are not supported.
- Lists: heterogeneous data structure with several fields. Each field is a matrix s.l. and these matrices (scalars, vectors and/or 2-dim matrix) can be different in size, dimension, and type. Each field has a name.

[] - Individual elements of an object (matrix or list) can be referenced by giving the name of the object followed by the subscripts in square brackets (separated by commas in the 2-dimensional matrix). E.g., vector[4], res1[2,4], res[,4] (i.e., all rows, column 4). Sub-ranges of a matrix can be referenced as, e.g., matrix[3:5, 4:10] (columns 3, 4, and 5, and rows 4 to 10)

$ - Individual variables of matrices and fields of lists can be referenced by giving the name of the data frame followed by $ and then by the variable name (the field). E.g: result$sp (the column sp from the object result); result$sp[3] (the specific value in position 3 of the sp column); [the usage of two $ (list$matrix$col) is not yet implemented, and thus, list$matrix[,1] should be used for the first column of a matrix in a list, and list$matrix[2,1] for the specific second value in col 1]

Optional ini file: **hist.ini** with commands to be loaded in the history buffer of the command line (these commands will be available with the key UP).

What's a landscape in lass?

A landscape is considered a matrix, i.e., a gridded landscape of squared cells. Numbers in the matrix may be the abundance of one species (e.g., 0: absence, 1:low, 2: med, 3: high) or the presence of a species in the cell (e.g., 0: empty cell; 1: cell with the species 1; 2: cell with species 2; etc.), and this would depend on the simulation model (library) used (e.g., FATELAND, MELCA, BROLLA). The `image` function is often used for visualising a matrix that will be used for generating landscapes. Usually, the function `landscape` generates the main parameters of the landscape (dimensions, cell size, wind properties); the function `distribution` is used for distributing the species in the gridded landscape. Functions like `matrix` and `pattern` can be used to generate specific landscape configurations that are then incorporated to the landscape object using the `distribution` function.

Examples of lass language usage:

```
# lass as a calculator
25 + (5 + 32)^2
a <- 25 + (5 + 32)^2
log(a)
```

Example: load MELCA model and execute (run) a simulation. Save the results in res_1 (matrix) and save the work space for further analysis

```
library(melca)
sp1 <- species.edit() # choose the species attributes from the species window
sp2 <- sp1
sp2$name <- "sp2"
sp2$resp <- "no"
land <- landscape(50,50)
pat <- pattern(matrix(0,50, 50), 1, "rand", 0.25)
pat <- pattern(pat, 2, "rand", 0.25, over=F)
land <- distribution(land, c("sp1", "sp1"), pat)
res_1 <- run(t= 100, land, plot=T)
plot(res_1)
names(res_1)
plot(res_1$t, matrix.col(sp1=res_1$mat1, sp2=res_1$mat2))
write(res_1, file=" run1.txt", header=TRUE) # save results in a text file
ls()          # list the variables in the worksapce
save.workspace("example1.lws") # save everything
rm.all()      # remove all variables everything
# all variables can be loaded again using:
load.workspace("example1.lws")
```

Libraries

Current available libraries are:

melca:

. A simple probabilistic spatially explicit model of vegetation dynamics (cellular automata) described in Pausas (2003).

. Available specific functions are: distribution, disturbance, landscape, species, run,

fateland:

. A spatially explicit version of the FATE model (by Moore & Noble 1990).

. Available specific functions are: climate, distribution, disturbance, landscape, species, species.disturbance, run,

brolla:

. A spatially explicit version of BROLLA, a gap-dynamics vegetation model developed by Pausas (1999).

. Available specific functions are: distribution, disturbance, landscape, species, run,...

Commands (functions) – by subject

general: ?, c, function, getwd, help, history, language, library, list, ls, load.workspace, names, rm, read, save.workspace, seq, setwd, source, str, write,

graphic: plot, image,

mathematics: abs, colmeans, colsd, exp, log, max, mean, min, rnorm, rowmeans, rowsd, matrixmean, matrixsd, round, sqrt, sum, table,

data management: list, mat2xyz, matrix, matrix.col, matrix.row, rep, replace, seq, setnames, subset, xyz2mat,

landscape: autocor, frags, pattern, topo,

simulation and random: library, random numbers, rnorm, rpois, rbinom, runif, rexp, set.seed,

specific functions for simulation models: see also libraries

distribution [melca, fateland, brolla]
disturbance [melca, fateland, brolla]
landscape [melca, fateland, brolla]
run [melca, fateland, brolla]
species, species.edit [melca, fateland, brolla]
species.disturbance [fateland]

Global commands (functions) – Alphabetic list

?, abs, autocor, c, colmeans, change, exp, function, frags, getwd, help, history, image, language, library, list, load.workspace, log, ls, mat2xyz, matrix, matrix.col, matrix.raw, max, mean, min, names, pattern, plot, random numbers, rbinom, read, rep, replace, rexp, rm, rnorm, rpois, runif, round, rowmeans, save.workspace, sd, setnames, set.seed, setwd, seq, source, sqrt, str, subset, sum, table, topo, write, xyz2mat.

autocor

global – A set of functions to compute spatial autocorrelation indices for testing spatial dependencies:

- `autocor.jc` compute global join-count statistics (BB and BW tests) for categorical data in regular matrices (gridded landscapes). Numbers in the matrix are assumed to be classes/categories (e.g., colors, species, types), and any number of classes is possible (currently it is only tested for matrices with 2 classes). BB or WW-tests assess positive spatial autocorrelation and BW-tests for negative spatial autocorrelation. The output provides for each adjancency pair, the number of adjacencies (`njoints`), the z-statistic (based on the randomisation assumption) and its variance, and the expected number of joins (xx?). If z is in the range of –1.96 and 1.96 (the 95% confidence interval in a Normal distribution), then the value is not extreme and likely due to chance (random distribution).
- `autocor.mg` compute global Moran's I, Geary's C indices (for quantitiative data) for regular matrices assuming that cells are adjacent (hence neighbourhood/adjacency is needed). Significant positive values reflect aggregation, while significant negative values reflect uniform distribution.
- `autocor.mgxyz` compute global Moran's I, Geary's C indices based on the Euclidean distance from xyz coordinates data (irregularly distributed data); hence lag is needed. If several lag distances are included, then the results can be used for producing spatial correlograms. Espheric coordinates (lat and long) should not be used here.

The statistics provided by `autocor.jc, autocor.mg`, and `autocor.mgeu` can be tested by Monte Carlo simulations (permutations) using `nsim` > 0. For each permutation a random distribution of the original matrix is performed and the index (join counts, Moran's I or Geary's C) is computed. Then it provides a Monte Carlo estimation of the p-value; in this case, all results for each permutation are optionally outputted (`showall`=T), and thus the user can employ these values as desired. Mean and standard deviation of the index for each permutation is also provided.

Description of the Moran and Geary indices is given in Cliff & Ord (1981); description of Gi and Gi* in Ord & Getis (1995) and Getis & Ord (1998)..

autocor.jc(m, neighb, nsim=0, showall=F)
autocor.mg(m, neighb, nsim=0, showall=F)
autocor.mgxyz(xyz, lag=c(1,1), cumulative=F, nsim=0, showall=F)

m – the matrix representing a gridded landscape to evaluate, ...
xyz – data to analyse in format of a matrix with 3 columns, euclidean coordinates x, y and the value. It is useful for irregularly distributed data.
neighb – neighbourhood (adjacency or contiguity) for regular gridded landscapes; options are "q" or "8" (8-cells, queen), "r" or "4" (4 adjacent cells, rook) and "b" (4 corner cells, bishop).
lag – a vector with 2 elements, lag distance and number of lags. Default lag= c(1,1).
cumulative – A boolean parameter to set whether I and C are computed for each lag exclusively (FALSE, default) or cumulatively (i.e., considering the data of the previous lags, TRUE).

nsim – number of permutations for computing the Monte-Carlo p-value (under the random hypothesis). If nsim >0, Monte-Carlo tests are executed and a p-value provided. If p>0.05, then the autocorrelation is not significantly different from the null hypothesis (random). If nsim=0, then permutations are not executed and the p-value is not provided.

showall – boolean. If nsim > 0 and showall=T, then, the results of the permuted matrix are saved. Default showall=F (not saved).

Output for autocr.mg(): a list with the following fields

lags – a matrix with 2 columns: lag (lag distance), npairs (number of neighbour pairs)

moran – a matrix with the following columns: moranI, znormI, varnormI, zrandI, varrandI. The number of rows are the number of lags requested.

lag: lag distance
moranI: Moran's I index
znormI, varnromI: a vector with the expected I under Normal assumption of z, and the variance
zrandI, varrandI: a vector with the expected I under Random assumption of z, and the variance

geary – a matrix with the following columns: gearyC, znormC, varnormC, zrandC, varrandC.

MCpI – (if nsim>0) Monte-Carlo p-value for I

MCpC– (if nsim>0) Monte-Carlo p-value for C

permutI – (if nsim>0 and showall=T) all permuted I values of the original data.

permutC - (if nsim>0 and showall=T) all permuted C values of the original data.

Examples:

```
m1 <- matrix(c(0,1), 10, 10)
autocor.jc(m1, nsim= 1000) # not random; 0/0 and 1/1 negative spatial
   correlation; 0/1 positive

m2 <- matrix(0, 10, 10)
m2[1:5, 1:5] <- 1
autocor.jc(m2, nsim= 1000) # not random; ; 0/0 and 1/1 positive
   spatial correlation (agregation); 0/1 negative

m3 <- matrix(0, 10, 10)
m3 <- pattern(m3, 1, "rand", 0.5)
autocor.jc(m3, nsim= 1000) # random

m4 <- m1+m2+m3
autocor.mg(m4, nsim=1000) # random

## Tricky example - the effect of the neighbourhood
m1<- matrix(c(1,0), 10, 10)
m11<- matrix(c(1,0), 9, 9)
autocor.jc(m1, neighb="8", nsim= 1000)
autocor.jc(m1, neighb="4", nsim= 1000)
autocor.jc(m11, neighb="8", nsim= 1000)
autocor.jc(m11, neighb="4", nsim= 1000)
##

m <- matrix(rnorm(100, 2, 2), 10, 10)
# convert matrix to coordinates:
m2 <- matrix.col(row= rep(seq(1,10), rep(10,10)), col=rep(seq(1,10),
   10), v= c(m))
```

```
# or a much easier way:
m2 <- mat2xyz(m)
autocor.mgxyz(m2, c(1,1))
```

c

Global – Combines the arguments to form a vector. If the argument is a matrix, it converts it to a vector by columns (concatenating the columns).

c(arg1, arg2, ...)

arg1, arg2, ... – objects of the same type to be concatenated in a vector

Examples:

```
a <- c(10, 5, 2)
b <- c("a", "b", "c")
c <- rnorm(5, 2, 1)
d <- c(a, c)

m <- matrix(rnorm(25, 1, 2), 5, 5)
v <- c(m)
m2 <- matrix.col(row= rep(seq(1,5), rep(5,5)), col=rep(seq(1,5),
   5),v=v)
```

function

Global – this is a special command to generate user-defined functions.
[to be completed]

Examples:

```
# simple example
negexp <- function(x) { g<- 0.0415; 1000*g^2 * exp(-g*x)/(2*3.1416)}

ss <- seq(1, 50)
plot(ss, negexp(ss))

# a more complex example:
```

frags

Global – compute indices to quantify landscape patterns. This function is based on FRAGSTAT version 2 (McGarigal & Marks 1994), with some modifications, and the indices are called as in FRAGSTAT 2 . Full definition of the indices and other details can be found in McGarigal & Marks (1994). The function generates a matrix or a list (with 2 or 3 matrices), with the name of the indices as column names, and the corresponding index value. Currently it computes 44 landscape, 38 classes and 12 patch indices. Different values (integers) of the input matrix are different classes. Patches are contiguous groups of cells of the same class and 3 neighbourhood options are possible (queen, rook, bishop). Fragstat allows for representing borders and background; then external patches of a given class have the same value as the internal patches of the same class but in negative form.

frags(m, cellsize, type= c("land", "class"), backgr, neighb="8", edgedist=0, weight, bweight, proxdist= -1, matrix=F)

m – input matrix of integers representing the landscape to analyse. Negative values represent external patches (outside the border of the landscape of interest).
cellsize – cell size (edge length) in meters
type – the type of metrics computed, that is, landscape, classes or/and patch indices ("land", "class", "patch", "matrix"). By default, it computes both class and landscape indices type = c("land", "class"). If the option "matrix" is included, it generates a matrix of the same size as the input matrix but with all patches numbered.
backgr – (optional) the absolute value of background code (an integer; that is, a positive value representing background). The positive form of this value will refer to interior background patches, and the negative to external background patches. If not included, then all values (including 0) are considered classes except -999 for exterior background and 999 or -990 for interior background.
neighb – neighbourhood (or contiguity); options are "q" or "8" (8-cells, queen), "r" or "4" (4 adjacent cells, rook) and "b" (4 corner cells, bishop). Default = "8".
edgedist – value (real) defining the edge distance [xxxx....]. Default = 0 (no differentiation between edge and core)
weight – (optional) a matrix with 3 columns defining weights for each class interaction [contrasts between classes xxx....]. If not included, then contrast indices are not computed.
bweight – (optional) bound weight, a value (real) defining the weight for the boundary-background
proxdist – search distance (in m) for the proximity indices. If not provided (or it is =<0) then proximity indices are not computed.

List of indices currently implemented:
landscape indices: TA, LPI, NP, PD, MPS, PSSD, PSCV, TE, ED, CWED, TECI, MECI, AWMECI, LSI, MSI, AWMSI, DFLD, MPFD, AWMPFD, TCA, NCA, CAD, MCA1, CASD1, CACV1, MCA2, CASD2, CACV2, TCAI, MCAI, MNN, NNSD, NNCV, MPI, SHDI, SIDI, MSIDI, PR, PRD, SHEI, SIEI, MSIEI, IJI, CONTAG.
class indices: TYPE, CA, PLAND, LPI, NP, PD, MPS, PSSD, PSCV, TE, ED, CWED, TECI, MECI, AWMECI, LSI, MSI, AWMSI, DFLD, MPFD, AWMPFD, CLAND, TCA, NCA, CAD, MCA1, CASD1, CACV1, MCA2, CASD2, CACV2, TCAI, MCAI, MNN, NNSD, NNCV, MPI, IJI.
patch indices: PID (patch identifier), TYPE, AREA, PERIM, EDGECON, SHAPE, FRACT, CORE, NCORE, CAI, NEAR, PROXIM.

Names and units of Frags indices:

TYPE	Patch Type
CA	Class Area (ha)
TA	Total Area (ha)
%LAND	Percent of Landscape (%)
LPI	Largest Patch Index (%)
NP	Number Patches
PD	Patch Density (#/100 ha)
MPS	Mean Patch Size (ha)

PSSD	Patch Size SD (ha)
PSCV	Patch Size CV (%)
TE	Total Edge (m)
ED	Edge Den (m/ha)
CWED	Con-Weight Edge Den (m/ha)
TECI	Total Edge Contrast (%)
MECI	Mean Edge Contrast (%)
AWMECI	Area-Wt Mean Edge Con(%)
LSI	Landscape Shape Index
MSI	Mean Shape Index
AWMSI	Area-Weighted Mean Shape
DLFD	Double Log Fractal
MPFD	Mean Patch Fractal Dimension
AWMPFD	Area-Weighted Mean Fractal
C%LAND	Core % of Landscape (%)
TCA	Total Core Area (ha)
NCA	Number Core Areas
CAD	Core Area Den (#/100 ha)
MCA1	Mean Core Area 1 (ha)
CASD1	Core Area SD 1 (ha)
CACV1	Core Area CV 1 (%)
MCA2	Mean Core Area 2 (ha)
CASD2	Core Area SD 2 (ha)
CACV2	Core Area CV 2 (%)
TCAI	Total Core Area Index (%)
MCAI	Mean Core Area Index (%)
MNN	Mean Nearest-Neighbor Distance (m)
NNSD	Nearest Neighbor SD (m)
NNCV	Nearest Neighbor CV (%)
MPI	Mean Prox Index
IJI	Intersper/Juxtapos (%)

Examples:

```
m <- matrix(0, 20, 20)
m <- pattern(m, 1, "hier", matrix.row(c(5,5,0.5), c(5,5,0.9)))
m <- pattern(m, 2, "rand", 0.5, over=F)
image(m)
fm <- frags(m, 1, backgr= 9) # no 9, no background
fm$land
fm$class
fm2 <- frags(m, type=”matrix”)
image(fm2)
table(fm2) # number of cells in each patch
```

getwd, setwd

Global - getwd return path representing the current working directory; setwd(dir) is used to set the working directory to dir.

getwd( )
setwd(dir)

dir – a character string indicating the path of the working directory.

Example:

```
wd1 <- getwd()
setwd("c:\lass\my models")
setwd(wd1)
```

help, ?

Global – provides access to documentation on a function (command)

help(function) #
function(?) # short and quick help for the function

Examples:
```
help(rnorm)
rnorm(?)
```

history

global – opens a text file with a list of the commands in the history.

history()

image

global – creates a coloured grid to visualise matrices.
pattern.edit( ) is available in the image menu.

image(m, col ="grey", range, min, reverse=F)

m - a matrix corresponding to the grid, or a matrix with the x, y, and z
col – colour options are: "grey", "red", "green", "blue", "terrain", "all". All colour options produce gradients of colour, except the option "all" in which each value has different colour in a non-gradient manner.
range – (as in `table`)
min – (as in `table`)
reverse – if TRUE then colours are selected in reverse order.

Examples:
```
image( matrix(seq(1,10), 10, 10), terrain )
image( matrix(c(0,1), 50, 50) )
image( t(matrix(c(0,1), 50, 50)) )
```

language, language.get, langmodel

Global – Allows changing the language of the menus and messages of lass and the associated libraries. `language` allows you to change the language of all lass messages

and loaded libraries. `language.get` generates an ascii file (similar to an .ini file), with all texts in the menus, windows and all error messages of lass and loaded libraries. `langmod` functions are the same than `language` functions but specific only for the loaded library. The default language in all LASS is english ("en").

language(lan)
language(file)
language.get(fileget)

langmodel(lan)
langmodel(file)
langmodel.get(fileget)

lan – indicates the language of the main menus and messages. Currently, the available options are: "`en`" (English) and "`es`" (Spanish). Other languages can be incorporated by the user using the `language(file)` and `language.load` options.
file - name (and path) of the input file (quoted, i.e. "") with all text in the language desired.
fileget – name (and path) of the file (quoted) with all text in the current language. This is useful for getting all text, translating it to the language desired and then loading it to lass using `language`(file).

Examples:

```
language.get("c:\lass\original.txt") # save lass text to a file
# it is possible to translate all text manually, for instance, to French, and then load it:
language("c:\lass\myfrench.txt") # load my version of lass texts
language("en") # return to the default language (English)
language("es") # set to Spanish
```

library

Global – Load or close a lass library. Loading any library closes any existing loaded library; that is, only one library can be available at any time.

library(name)
library()

name – string giving the name of the library to be loaded. If name is missing, then it closes the loaded library (if any).

Example:
```
library(fateland)
```

list

Global – for constructing a list (heterogeneous lass data structure)

list(arg1, arg2, ...)

arg1, arg2, ... – objects to be included in the list (matrices, vectors, numbers...).

Example:
```
mylist <- list(a= rnorm(5,5,5), c= "hola", d= c("a","b","c"))
mean(mylist$a)
# new fields can be included simply by:
mylist$ee <- "new"
# fields can be removed by assigning the field as empty:
rm.field(mylist, ee)
```

ls

Global – return a matrix of character strings giving the names of the objects in the current environment.

ls()

Example:
```
ls()
a <- ls()
a
```

mean, sd, max, min, sum, summary

global – compute the maximum, the minimum, the sum, the mean and the standard deviation of a numeric object (vector or a matrix). summary computes all 5 values and also provides the number of values of the object.

min(a1, a2, ...)
max(a1, a2, ...)

mean(a1, a2, ...)
sd(a1, a2, ...)

sum(a1, a2, ...)

summary(a1, a2, ...)

a1, a2, ... – numbers or numerical objects

Examples:

```
a <- rnorm(100, 5, sd= 5)
min(a)
summary(a)
a <- matrix(a, 10, 10)
min(a[, 1])
sum(a)/sum(table(a)) # mean
mean(a)
```

mean, sd, colmeans, rowmeans, colsd, rowsd, matrixmean, matrixsd

Global – Compute the column or row means and sd for numeric arrays (vectors, matrices).

mean(a1, a2, ...)
sd(a1, a2, ...)
provides a single value of mean/sd for all data in the objects

rowmeans(obj1, obj2, ...)
colmeans(obj1, obj2, ...)
rowsd(obj1, obj2, ...)
colsd(obj1, obj2, ...)
provides rowmeans/sd and colmeans/sd for matrices

matrixmean(m1, m2, ...)
matrixsd(m1, m2, ...)
provides a matrix with means value at each position of the matrix (the mean of that position thoughout all matrices)

a1, a2, ... – numbers or numerical objects
obj1, obj2, ... – numerical objects of the same number of rows. If more than one matrix is provided, mean and sd are computed as concatenated rows (rowmeans, rowsd) or concatenated columns (colmeans, colsd). Note that colmeans and rowmeans are functions for matrix, but if one or several vectors are provided, vectors are assumed to be rowa, and so, colmeans and rowmeans give different values for a vector (see examples.
m1, m2, ... – numerical matrices of the same dimensions. `matrixmean()` and `matrixsd()` – produce a matrix of the same dimensions to m1, m2, ...

Examples:

```
m <- matrix( seq(1, 15), 5, 5)
colmeans(m)     # means of all rows for each column (a vector)
mean(colmeans(m))   # mean (a value) of the vector above
rowmeans(colmeans(m)) # the same
mean(m)     # the mean (a value) of all numbers in the matrix m

rowmeans(m[,2:5])     # row mean of columns 2, 3, 4, and 5
m2 <- m1
rowmeans(m, m2) # a vector with row mean of all columns in the 2 matrices (not mean of 2 means!)
m2 <- 2*m
matrixmeans(m, m2) # a matrix with the means of each position

a <- seq(1, 5)
mean(a) # mean of the numbers: 3
colmeans(a) # a is assumed as 1 row, colmeans(a) = a
rowmeans(a) # a is assumed as 1 row, rowmeans(a) = 3
colmeans(a, a) # a
rowmeans(a, a) # 3
```

names

Global – shows the names of the fields (in a list) or the names of the columns (matrix)

names(obj)

Example:

```
mylist <- list(a= rnorm(5,5,5), c= "hola", d= c("a","b","c"))
names(mylist)
mymatrix <- matrix.col(seq(0,5), seq(10,15))
names(mymatrix) # no names
mymatrix <- matrix.col(a= seq(0,5), b= seq(10,15))
names(mymatrix)
mymatrix$a
```

matrix, matrix.col, matrix.row

Global – generates a matrix from a vector, or from several vectors by columns or by rows. `matrix.col()` allows you to include matrix column names (in lass, matrix row names are not supported).

matrix(v, nr, nc, byrow=F)

matrix.col(col1, col2, ...)
matrix.row(row1, row2, ...)

col1, col2, ... *row1, row2*, ... – are vectors of the same length and type (numbers or strings) to be converted to columns or rows of a matrix

v – a value or a vector. If v is a number, then a matrix of dim nr x nc is created with all values equals to v. Otherwise, if v is a vector, a matrix of nr x nc is created using the vector (by rows (default) or by columns); in this case, if the vector is shorter than nr x nc, it is recycled, if the vector is longer than nr x nc, it is cut

nr, nc – number of rows and columns

byrow – whether the vector (v) is set to the matrix by columns (default, byrow=F) or by rows.

```
mymatrix <- matrix.col(a= seq(0,5), b= seq(10,15))
x <- c(1, 2, 3, 4)
y <- c(10, 20, 30, 40)
xy <- matrix.col(x=x, y=y)
xy2 <- matrix.row(x, y)  # this will produce the same as: t(xy)
mat <- matrix.col(rnorm(100, 2, 2), rnorm(100, 2, 4))
matrix(0, 10, 10)
matrix(c(1,2,3), 2, 2)
# 1 3
# 2 1
matrix(c(1,2,3), 2, 2, byrow=T)
# 1 2
# 3 1
matrix( seq(1, 25), 2, 2)
# 1 3
# 2 4

# matrix to coordinates - see also mat2xyz()
```

```
m <- matrix(seq(1, 25), 5, 5)
m2 <- matrix.col(row= rep(seq(1,5), rep(5,5)), col=rep(seq(1,5),
   5),v=c(m))
m3 <- matrix.col(x= rep(seq(1,5), 5), y= rep(seq(5,1), rep(5,5)), z=
   c(m))

# pattern generation using matrix
m <- matrix(0, 10, 10)
m[1:5, 1:5] <- 1
m[6:10, 6:10] <- 2

x <- rep(seq(1, 50, by= 2), 50)
y <- rep(seq(1, 50), rep(25, 50))
vert <- matrix.col(x = x, y= y) # vertical stripes
hor <- matrix.col(x = y, y= x) # horizontal stripes
```

even simpler (see also the option "short" of the function pattern):

```
hor <- matrix(c(0,1), 50, 50) # horizontal stripes
hor2 <- matrix(c(0,1,0,0,0), 50, 50) # thinner stripes
obl <- matrix(c(0,1,0), 50, 50) # oblique
ver <- t(hor)
```

pattern, pattern.edit

global – fills and produces patterns in a given matrix. Often used for generating patterns of species distribution in the landscapes or for disturbance patterns. Some patterns can be generated directly by `matrix` (and may not need `pattern`). `pattern.edit()` is a GUI version of pattern function and it is also included in the top bar of the image function (click on the button indicated with a "Pattern"). It permits the visualisation of the matrix and also permits generating and/or modifying matrices using the mouse. The options are basically the same but friendlier.

pattern(m, val, method, par, over=T)

`m` – a matrix. If m is a subrange of a matrix, any pattern would only affect the subrange.
`val` – a value (a number without quotes) or string (with quotes)
`method` – method used to include the val in the appropriate location of the matrix. If not included, then all cells of the matrix are filled with the value in val. Available methods are:

- "rand" – uniform random positions
- "hier" – hierarchically structured random pattern
- "coord" – fixed position based on row (y), col (x) co-ordinates
- "gradient" – horizontal, vertical or oblique (combining horizontal and vertical) gradient of `val`.
- "matrix" – positions are chosen from a matrix. Non-zero values of the matrix in `par` are the position where the value 'val' is inserted in the matrix 'm'.
- "short" – short way to describe matrix patterns

`par` – parameter associated with the method. Required if method is included

if "rand" then a probability value (0 to 1)
if "hier" then a matrix with the x and y levels and the probabilities (in this order: first, second and third column) for hierarchically structured random landscapes.
if "coord" then a matrix with 2 columns, for rows (y) (1st col) and columns (x) (2nd col). Or a matrix with 3 columns where the 3rd column is the value

to be included in its corresponding place (in this case, the parameter val is ignored, although required).

if "gradient" then a vector of up to 4 values. The first and the second are the slopes of the vertical and horizontal gradients. If only one value is provided, only vertical gradient is assumed (horizontal slope = 0). Slopes can be negative. Oblique patterns can be generated by setting both first and second values different to 0. The third and fourth elements are values from 0 to 1 for the relative position of the gradient line (default= 0.5 for both horizontal and vertical, i.e., the 0.5 probability is in the middle).

if "matrix" then par is a matrix. Non-zero values of the matrix in par are the position where the 'val' is inserted in the matrix m.

if "short" then par is a matrix (or vector) showing the pattern, and this pattern will be redimensioned. E.g., `par= c(0,1)` would fill the right 50% of any matrix; `par= c(0,1,0,1,0,1,0)` would produce vertical stripes in any matrix; etc.

`over` – overwrite the non-zero values of matrix m

Examples:

```
mat <- matrix(0, 10, 10)
mat <- pattern(mat[1:5, 1:5], 1, "rand", 0.3)
mat <- pattern(mat[6:10, 6:10], 2, "rand", 0.8)

pattern(mat, 0, "coord", matrix.col(a=c(1,2,3), b=c(1,1,1),
v=c(50,15,35))) # the val 0 is ignored

# for prob=1, it can be done directly by filling submatrices with 1
mat <- matrix(0, 10, 10)
mat[1:5, 1:5] <- 1
mat[6:10, 6:10] <- 2
# substitution of specific locations using the option matrix:
mat2 <- pattern( mat, 5, "matrix", matrix.col(c(0,1), c(1,0)) )

mat <- matrix(0, 100, 100)
image(pattern(mat, 1, "short", par= c(0,1,0,1,0 )) ) # vertical lines

# hierarchically structured random pattern
mat <- matrix(0, 10, 10) # a matrix with all 0
hmat1 <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(1,0.5,0.9) )
mat <- pattern(mat, 1, "hier", hmat1)
mat <- pattern(mat, 2, "rand", par=1, over=F) # fill the rest with "2"
# would be similar to:
mat <- matrix(2, 10, 10) # a matrix with all 2
mat <- pattern(mat, 1, "hier", hmat1, over=T)

image(pattern(matrix(rnorm(10000,1,10),100,100), 1, "hier", hmat1))
```

plot

global – for plotting lass variables (x, y). For matrices, columns are ploted as y values. In this function single vectors are considered as columns.

plot(x, y, type="l")
plot(m, type="l")

plot.y(m2, type="l")

x - a vector
y – a vector or a matrix of vectors of the same length as x
m – a numerical matrix. The first column is considered x, and the other columns are different series of y.
m2 - a numerical matrix (or single column vector). All columns are considered as y and the x axis is a sequence from 1 to the number of rows.
type – "p" (points), "l" (lines)

Example:

```
t <- seq(1, 50)
y1 <- rnorm(50, 5, 1)
y2 <- rnorm(50, 5, 2)
plot(t, matrix.col(y1, y2))
plot.y(matrix.col(y1, y2))  # the same as previous
```

rm

Global – remove objects from the workspace.

rm(obj1, obj2, ...)

obj1, obj2 – are the lass objects to be removed from the shell

Example:

```
t <- seq(1, 50)
```
rm(t)

read / write

read reads an external ascii file (by rows) and converts it to a lass object.
write writes an external ascii file from a lass object (x).

read(file, header= F, sep = "", dec=".", thousand ="", skip=0, comment ="#", skipblank= T)
write(x, file, header= F, sep=" ", dec=".", digits= 3)

file – file name (and path, if different from the current). E.g., file="c:\lass\dat.txt"
x – a lass object
header – column names
sep – field separator. In read, if sep = "" (default) the separator is one or more spaces or tabs. In write, sep="" means no separation between columns.
dec – decimal character. Default = "."
thousand – character separating thousands. Default = no character used.
skip – number of lines skipped (not read)
comment – character that define comments. Default = "#"

skipblank – Boolean; if TRUE, where is there is a blank line, it is skipped (not considered); otherwise, a blank line causes an error.
digits – number of digits used when writing in the file

Example

```
m <- matrix(rnorm(100, 2, 2), 10, 10)
write(m, "tmp.txt")
m2 <- read("tmp.txt")
```

rep

global - replicates the values in x according to the values given in times

rep(x, times)

x – a vector of any type and length
times - integer. A vector giving the number of times to repeat each element of x

Examples:

```
rep(seq(1,3), 3)    # 1,2,3,1,2,3,1,2,3
rep(seq(1,3), c(3,3,3)) # 1,1,1,2,2,2,3,3,3
rep(seq(1,3), rep(3,3)) # 1,1,1,2,2,2,3,3,3

m <- matrix(rnorm(25, 1, 2), 5, 5)
m2 <- matrix.col(row= rep(seq(1,5), rep(5,5)), col=rep(seq(1,5),
   5),v=v)
```

replace

Generates an object with the same structure as the input object 'm', but with the values in "in" replaced with the values in "out". Optionally, 'else' is also possible. 'm' remains unchanged.

replace(m, in, out, else=NULL)

m – a vector or a matrix
in – a value or a vector of values in matrix `m`
out – a value or a vector of the same length as `in`, with the resulting values
else – optional. A value. If included, all values in m that are not in "`in`" are changed to the value else. By default (else not included), values not included in "`in`" are not modified.

Examples:

```
a <- seq(1, 10)
b<- replace(a, 10, 1000) # replace 10 by 1000
c <- matrix(a, 10, 10)
d <- replace(c, c(1,2,3), c(0,0,0), else=1)

# another example: analysing simulation results
res <- run(100, land, save2="spp" ) # fateland
table(res_Erica_100)
```

```
# convert to presence/absence
#  note that 0 needs to be included (and repalced by 0) if "else" is considered
in <- c(-4,-3,-2,-1, 0) # seeds or absences
ou <- c( 0, 0, 0, 0, 0)
table(replace(res_Erica_100, in, ou, else= 1)) # presence/absence
```

random numbers: rnorm, norm, rbinom, rpois, rexp, runif

Global – Random generation numbers with different distributions (Normal, Binomial, Poison, Exponential, and Uniform). Random generation for the normal distribution with mean equal to `mean' and standard deviation equal to `sd'. norm( ) is a short version of rnorm(1, mean, sd).

rnorm(n, mean= 0, sd = 1)
norm(mean, sd =1)

runif(n, min= 0, max=1)
rbinom(n, size, prop)
rpois(n, lambda)
rexp(n, rate= 1)

n - number of observations to be generated
mean - mean value (def. = 0)
sd - standard deviation (def. = 1)
min – minimum value of a uniform distribution (def = 0)
max – maximum value of a uniform distribution (def = 1)
prop - probability of success on each trial. Default = 0.5
lambda – Poisson parameter (mean)
rate – rate for the Exponential distribution (1/rate)

Examples:

```
mean(rnorm(100, 5, 3))
table(rbinom(100, 5, 0.2))
table(runif(100, 1, 5), 1)
matrix.col(a= rnorm(5,5), b= rbinom(5, 10, 0.5), c= rpois(5,10))
```

round, abs, sqrt, log, exp

global – Some standard function for rounding the number of decimals, for obtaining absolute value and for standard mathematical operations.

round(x, d)
abs(x)
sqrt(x)
log(x, base)
exp(x)

x – a numeric object (a value, a vector, a matrix or a subrange of a matrix).
d - integer indicating the precision to be used in rounding (default 0)

base – the base with respect to which logarithms are computed. It must be a positive number. [not yet implemented (base = e assumed)]

Examples:

```
a <- seq(-2.5, 4.5)
round(a)
```

save.workspace, load. workspace

Global – used for saving all variables in the work space to a file; loading the previously saved workspace.

save.workspace("filename")
load.workspace("file")

set.seed

Global – for setting the seed value for the random number generator

set.seed(n)

n – optional integer value; if not provided, then seed is taken randomly (randomise)

Examples:

```
set.seed(123)
a <- rnorm(5, 3, 2)
set.seed()
b <- rnorm(5, 3, 2)
set.seed(123)
c <- rnorm(5, 3, 2) # equal to a, different from b
```

setnames

Global - Set (or substitute) the column names of an object (data frame or list)

setnames(obj, v)

obj - object (data frame or list)
v - vector of strings with the column names desired

The number of columns in *m* must be the same as the number of strings in *v*

Examples:

```
ve <- c("col1", "col2", "col3")
m <- matrix( rnorm(15, 3, 1), 5, 3 ) # m has 3 columns
setnames(m, ve)
setnames(m[,3], c("c")
setnames(m[,1:2], c("a", "b"))
```

seq

Global – generate vectors that are sequences ....

seq(from, to, by=1)

from – starting value of sequence
to - maximal value of the sequence
by - increment of the sequence

Examples

```
m <- matrix(seq(1, 25), 5, 5)
m2 <- matrix.col(row= rep(seq(1,5), rep(5,5)), col=rep(seq(1,5),
    5),v=c(m))
```

size

global – size (dimensions) of a matrix (or vector) or the number of fields of a list. Note that in lass, a vector is, in fact, as a matrix of one row.

source

global - source causes lass to accept its input from the named file (the name must be quoted). Input is read from that file until the end of the file is reached. `source` admits in the file variable names with %. In this case, the result is the character % been substituted by the current value of `ntimes` (see example). `source` cannot be nested, i.e., the function `source` cannot be called from the input file. The execution of the file can be canceled with Ctrl-c.

source("file", ntimes= 1, echo=F, ask=F)

file – name (and path, if needed) of the input file with lass commands
ntimes – number of times to execute the file
echo - logical; if TRUE, each expression is printed before evaluation.
ask – logical; if TRUE, each expression is printed before evaluation (as in echo=T), then it stops at each line, just before executing. The user is asked to press return (for confirmation) before executing. Pressing ESC skips the line. The 'echo' option is ignored (and assumed as TRUE).

Example:

file: test.txt

```
-------------------------------------------------------------------
land% <- pattern(m, 1, "rand", 0.3)
res[%] <- mean(land%)
-------------------------- endfile --------------------------------
```

lass:

```
m <- matrix(0, 100, 100)
res <- c(0,0,0)
source("c:\lass\test.txt", 3)
res
```

`#` will generate 3 variables (matrices): `land1, land2,` and `land3` -try `ls()-,` and fill the vector `res` with the results of `mean` (the mean of each new matrix).

str

global – Displays a summary (the structure) of a lass object (dimensions, type, and first values). For lists, displays the structure and the first values of each field. It is a useful way to visualise lists.

str(obj)

obj – any lass object (vector, matrix, list)

Example:

```
aaa <-list(a= rnorm(22, 1, 5), b= rep(2, 10), c="hola", d=matrix(1,
10, 10))
str(aa)
```

subset

Global - Return subsets of matrix defined by the row numbers and column numbers

subset(m, rows, cols)

m – a matrix
rows, cols – a vector defining the rows/cols to be selected

Example:

```
m <- matrix(seq(1, 25), 5, 5)
subset(m, cols=c(1,3,5))
```

t

Global – transpose matrices (or vectors)

t(m)

m – a matrix

Example:

```
y1 <- seq(1,10)
y1
t(y1)
y2 <- rnorm(10, 5, 2)
plot( t(y1), t(y2) )
```

table

Global – Frequency table (table of counts)

table(var, range = 0, min = 0)

var – a numerical vector
range – range of the class level, size of the level
min – a specific value that will be the lower limit of a range

Examples:

```
a <- pattern(matrix(0, 10, 10), 1, "rand", 0.5)
a <- pattern(a, 2, "rand", 0.3)
a <- pattern(a, 3, "rand", 0.1)
table(a)
table(a, range= 0.6)
table(a, range= 0.6, min= 2)

table( rbinom(100, 5, prob=0.3) )
table( rnorm(100, 10, sd=1), range= 2)
```

topo

Global – generate a matrix representing a topography, by interpolating from some points (row, column and altitude values) using the Cressman algorithm (Cressman 1959). The object generated is a matrix that can be visualised using `image`, and used as topography in FATELAND.

topo(altmatrix, nr, nc)

nr, nc – number of rows and columns
altmatrix – a matrix with 4 columns and as many rows as points included for the generation of the topography. Columns are: row, column, altitude, r (radius of influence for smoothing). Different radius can be used for each point, representing .....

Example:

```
alt <- matrix.col( r= rep(c(20, 50, 100), 4), c= c(rep(1, 3), rep(30,
    3), rep(60, 3), rep(90, 3)), alt= c(100,50,20, 10,10,10, 50,20,10,
    10,5,1), rad= rep(50, 12))
t <- topo(alt, 100, 100)
image(t)

alt[, 4] <- 20 # or alt$rad <- rep(20, 12) # changing the radius
image(topo(alt, 100, 100))

# some simple topographies can be generated easily without the topo function
library(fateland)
land <- landscape(....)
land$topo<- matrix(seq(1,100), 100, 100) # constant slope
```

xyz2mat, mat2xyz

Global – Change format, from a matrix to coordinates (row, col, value) and viceversa.

xyz2mat(rcv, over=F, defval=0)

mat2xyz(m, defval=0)

rcv – a matrix with 3 columns: the row and the col (the coordinates) and the value (in this order)
over –
defval – in mat2xyz, defval is the value that is not saved. In xyz2mat defval is the value that is included if the coordinates are not given
m – a matrix

Examples:

```
m1 <- matrix(seq(1, 25), 5, 5)
m2 <- mat2xyz(m1)
m3 <- xyz2mat(m2)
# m1 and m3 should be the same:
sum(m1-m3)  # should give 0

m4 <- subset(m2, rows=seq(6,25) ) #
xyz2mat(m4)
xyz2mat(m4, defval=-9)
```

MELCA – brief description

MELCA version 1 is described in Pausas (2003). Main modifications from this published version are:

- All cells may have seeds (including cells with immature or mature plants). Seedbank is a list with a maximum of 5 effective seeds per species. If a 6th seed arrives, then the oldest is removed.
- Annual probability of mortality and annual probability to mature is substituted by age to mature with SD and longevity with SD.
- Several errors fixed. kdisp was not used, etc.
- Competitive interactions: stratum (layers): 1 (Tree and Pinus), 2 (shrub) and 3 (herb).....
- Random fire (in time) set with with starting year. OldMelca set a fire in the starting year and then set the random regime. The NewMelca starts the random regime in the starting year
- Secondary juvenile period determined by the function: sjuv= 0.88*pjuv^0.47 (calculated form Pausas et al. databases (BelindaDB))
- Competitive interactions (substitution):
  Allowing a secondary species
  Existing species A versus new species B (seed or lateral growth). Who wins?
  Depends on the stratum and the shade tolerance:
  Strata: 1- Tree (Tree or Pinus); 2 – shrub; 3 – Herbs
  ShadeTolerance: Low, moderate, high

	Stratum		
ShadeTol	A < B	A = B	A > B
A < B	A	A	A
A = B	A	A	B
A > B	B	B	B

  A means that there is no substitution.
  B means that species B created a seedling-sapling (secondary species) that will substitute A if A dies or when B reaches the maturity age (delayed substitution to mimic competition).
- Fuel-driven fire is different.

Lass functions for Melca

distribution, disturbance, landscape, run, species,

distribution

melca – modify the object (list) generated by landscape by including the position and age of the species. That is, it generates the initial condition for running melca. If no distribution matrix is given or all values of the matrix are 0, then the species is removed from the landscape. A 1 will include the species in all the cells. Generates a list, equal to landscape given in land=, but with two new added fields: species (the names of the species) and distribution (a matrix with the following four columns (in this order: x, y, age, species).

distribution(land, sp, matrix, age = "b", over = F)

land – object generated by landscape.

sp – the name of a species object or vector with the list of the names of the species objects. Species objects should be in the workspace, generated by `species()`.

matrix – a matrix with the distribution of the species, where each cell corresponds to the location in the space. In each cell: 0: absence, 1: presence of the first species in sp, 2: presence of the second species in sp; etc. If the value is negative, then a static cell is included (static cells are empty cells that do not change with time, e.g. outside area, non-vegetation, etc.). If sp includes fewer species than the values of the matrix, then unmatched values are not considered (i.e., absence). This matrix is often generated with `matrix()` and/or `pattern()`.If the *matrix* and *land* paramenters have different dimensions, the *matrix* paramenter is not redimensioned (see `pattern` for redimensioning from small matrices), and the overlying section is used. Note that even if the matrix have several species, the run function allows for simulating a reduced set of species. If the *matrix and land paramenters have different dimensions, the matrix paramenter is not*

age – the age distribution of the plants. The method for selecting the age of each plant. Options are (default is age = "b"):

- "i" – all plants are immature, with age uniformly random (between 1 and `matureage` of the corresponding species);
- "m" – all plants are mature, with age uniformly random (between `matureage` and `maxage`);
- "b" – both mature and immature, with age uniformly random (between 1 and `maxage`)
- "r" – age distribution following a binomial distribution, which can be right- or left-skewed.
- "y" - population dominated by young plants;
- "o" - population dominated by old plants.
- "n" – normal distribution

over –whether the species matrix should overwrite any previous species matrix. If true, then the first matrix is overlayed with the second, that is, filled cells may be replaced, and empty cells may be filled.

agepar – a vector of the two parameters for the distribution function. If *sp* includes several species, then agepar is a matrix with the parameters in columns, of as many rows as the number of species included in sp (otherwise, that is, if only one vector with two parameters is included and there are several species, then it is assumed that all the species have the same parameters).

if age="r", then agepar is the probability of a binomial function. That is, if agepar>0.5 the age distribution is skewed to the left (adult plants dominance), and the opposite if agepar<0.5 (biased towards young plants). For agepar= 0.5, the age distribution follows a Normal distribution.

if age= "y", then the b and c of: probability = exp(- ((a/b)^c)); where a is a uniformly random value from 1 to maxage. Default values are b= maxage/2, and c= 2.

if age= "o", then the b and c of: probability = 1- exp(- ((a/b)^c)); where a is a uniformly random value from 1 to maxage. Default values are b= maxage/2, and c= 2.

if age= "n", then the mean and sd of the age (default: mean= maxage/2, sd= mean/10; i.e. CV= 10%)

Examples:

```
# sp1 with cover ca. 10% of the landscape:
myland <- distribution(landscape(50,50), sp1, pattern(matrix(0,50,
      50), 1, "rand", 0.1))
# produce the distribution matrix, with the x, y coordinates, the age and the species code
myland$species
names(land1$distribution)
myland$distribution[1:10, ]

# Another example. Generating a distribution directly setting x, y, age, and species:
dist <- matrix.col(x=rep(seq(1, 50, by= 2), 50), y=rep(seq(1, 50, 1),
      rep(25, 50) ), age= round(rnorm(1250, 10, 5)), sp=rep(c(1,2),
      625))
myland <- landscape(50, 50)
myland <- distribution(myland, sp=c("sp1", "sp2"))
myland$distribution <- dist


# stripes (banded landscapes):
stripes <- matrix(c(0,1), 50, 50)
image(stripes)

sp1 <-species("ph", ...)
sp2 <-species("qi", ...)
myland <- landscape(50, 50)
names(myland)
myland <- distribution(myland, sp=c("sp1", "sp2"), matrix= stripes)
names(myland)

# another simpler way to create stripes
p <- pattern(matrix(2,50,50), 1, "short", c(1,0,1,0,1,0,1,0))
myland <- distribution(landscape(50,50), sp=c("sp1", "sp2"), matrix=
      p)

# hierarchically structured random landscapes
hmat1 <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(1,0.5,0.9) )
hmat2 <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(0.5,1,1) )

pm <- pattern( matrix(0, 50, 50), 1, "hier", hmat1)
pm <- pattern(pm, 2, "hier", hmat2)

myland <- distribution(landscape(50,50), c("sp1", "sp2"), pm)

# another example:
mat <- matrix(2, 10, 10) # a matrix with all 2
mat <- pattern(mat, 1, "hier", hmat1, over=T)
myland <- distribution(landscape(50,50), c("sp1", "sp2"), mat,
    age="n")
```

in this case, the age distribution of the 2 species has the same parameters, and is dependent on the `maxage` parameter of each species.

disturbance, disturbance.edit

Melca – set the disturbance parameters for the Melca run by generating a list. `disturbance.edit()` is a friendly window interface version for `disturbance()`.

disturbance(tpar, ttype="rand", first = 1, sn = 1, stype = "rand", loc = NILL, around=T, ptype = "fuel", ppar)
disturbance.edit(distobj)

-- time parameters:
ttype – disturbance frequency type:
"rand": randomly
"syst": systematic
"weib": weibull distribution
tpar – disturbance frequency parameter.
if "rand" then the disturbance probability (0-1)
if "syst" the disturbance interval or a vector with the mean disturbance interval and the CV
if "weib" then average disturbance interval and the k coefficient for the Weibull distribution. if k= 1, then it is a negative exponential distribution (def: 2.3)
first – year in which the parameters function `disturbance()` is first applied

-- space parameters:
sn – number of disturbance events (an integer) or a vector with the means and the coefficient of variation (%) of the number of disturbance events in each disturbance year (def.: sn=1).
stype – how the disturbance initial cells (e.g., fire ignition cells) are chosen:
"rand": randomly
"syst": systematically; if stype = "syst", then loc is needed (e.g. fire ignition cells), and sn is not considered
"fuel": fuel random. Fuel-driven fire.
loc – a matrix of 2 columns (x and y coordinates) and as many rows as the number of initial disturbance points. sn is not considered (the number of initial points is the number of rows).
around – in case the initial cells are not burnable (empty or seeds only), whether the fire should start in the closest cell (def: around = yes). Available for `stype= "rand"` or `"syst"`. Note that if a coefficient of variation is given in `sn` and `around= FALSE`, then the annual number of disturbances is quite unpredictable, but most probably lower than the mean value of `sn`.

-- propagation parameters:
ptype – type of propagation:
"fuel" - fuel based, direct, simple (fire disturbance)
"prob" - based on a constant probability (0-1)
"radius" - disturbed everything within a square of radius (1/2 side) of ppar
"matrix" – disturbed pattern as in the matrix provided in `ppar`. In this case `sn` and `stype` are ignored.
"all" - all landscape is disturbed. In this case, `sn` and `stype` are ignored.
"hier" – hierarchically structured pattern. In this case, `sn` and `stype` are ignored.
ppar – propagation parameter:
if ptype = "prob", then probability (0-1)
if ptype = "radius", then length of the radius in number of cells

if ptype = "matrix", then a matrix of the same size as the landscape with 1 for the cells burned and 0 for the unburned cells (can be generated by `matrix` or `pattern`).
if ptype = "all", then `ppar` is ignored.
if ptype = "hier", then it is a matrix with the x and y levels and the probabilities (in this order: first, second and third column) for the hierarchically structured random landscapes.

distobj – a disturbance object generated with `disturbance()`.

Examples:

```
rfirehigh <- disturbance(0.05)
rfirelow <- disturbance(0.01)
# similar, but different fire regimes are:
rfirehigh2 <- disturbance("syst", tpar= c(20, 10))
rfirelow2 <- disturbance("syst", tpar= c(100, 10))

xy <- matrix(x= c(1, 2, 3, 4), y= c(10, 20, 30, 40))
rfire2 <- fire("rand", 0.05, stype= "syst", loc= xy)

fsys20all <- disturbance("sys", 20, ptype = "all") # everything disturbed
    every 20 yrs

# creating disturbance patterns
hmat <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(1,0.5,0.9) )
distmat <- pattern(matrix(0, 50, 50), 1, "hier", hmat)
mydist1 <- disturbance("rand", 0.05, ptype= "matrix", ppar= distmat)
mydist2 <- disturbance("rand", 0.05, ptype= "hier", ppar= hmat)


# both mydist1 and mydist2 are hierarchically random disturbance patterns; however the
are different:
# mydist1: in every disturbance year the cells disturbed are the same, and are given by
the matrix distmat
# mydist2:  hmat is the matrix providing the parameters for the hierarchical random
disturbance pattern, and so the exact cells disturbed are different each time
```

landscape

Melca – generates an object with the characteristics of the landscape

landscape(nr, nc, psize = 10, maxcover = 100, cv = 0, wd = NILL, wi = 0)

nr – number of rows of the gridded landscape
nc – number of columns of the gridded landscape
psize – pixel size in meters (def = 10) (currently, only square cells are acceptable)
maxcover – maximum cover (def = 100)
cv – coefficient of variation of the maximum cover
wd – direction of the dominant wind (def = NILL, no wind): "n", "ne", "se", "s", "sw", "w", "nw"
wi – intensity of the wind ("none", "low", "med", "high)

Example:

```
myland <- landscape(50, 50)
```

run

Melca – Run is the basic function to execute the Melca model and generate simulation results. Control-Q stops (aborts) run simulation processes.
If plot=T then it allows a friendly interface (GUI) for interactive simulations, including the dynamic visualisation of the landscape, a dynamic plot, possibility to save current results, pause/continue/stop buttons, etc. (see run(..., plot =T), next function)

run(t = 0, land, sp, dist, tstep=1, plot=F, grid= F, out= "b", save="final", savet=-1, savef=FALSE)

t – length of time to simulate. If t= 0, no simulation is done. Used when display = T for interactive simulations.
land – object generated by `distribution`
sp – (optional) A species object (generated by `species()` function) or a list (vector) of names of the species objects to be considered for the simulation. If not given, all species included in the land (defined by `landscape()`) are included
dist – (optional) object or list (vector) of names of the objects with the disturbance parameters generated by `disturbance()`. It is an optional parameter; if not provided then disturbance do not occur during the simulation. It permits multiple disturbances by including a list of disturbance objects.
tstep - time step for displaying and saving the results (default, tstep=1, annual time step).
plot – Shows a dynamic plot and grid during the simulation, and also all GUI facilities for run (see run(..., plot =T), next function).
out – (optional) output required: "s" – by species; "t" - "total" only, "b" both
tstep – (optional) time step to save results (def: tstep= 1, i.e. every year)
save – (optional) optional save maps (in the hard disk or in the environment, see savef):
"no": default
"final": save final landscape
"dist": disturbance pattern (it saves as many maps as disturbances)
"distrec": disturbance recurrence (a final map)
"map": save a map every X years as set in *savet*.
savet – is the year or sequence of years that the save should be executed. In the case of a sequence of years, these need to be sorted!. Negative values indicate not to save.
savef – whether the output is saved in a file (savef=TRUE) or to the assigned variable only (savet=FALSE). The file name will start with the name of the assigned variable, if any, or with "_".

Output:
`run` returns a matrix with the following variables:
- run: run number (if nruns> 1, otherwise this variable is omitted)
- t: time (years)
- distur: number of cells disturbed (if dist parameter is included)

Total and for each species:

- imm: number of cells with Immature (status= immature)
- mat: number of cells with Mature (status= mature)
- plsdl: number of cells with seedlings (status= mature and seedling)
- plseed: number of filled cells (immature or mature) with seeds
- seed: number of empty cells with seeds (seedbank only)
- seedt: total number of seeds (sum of the seeds of all cells)

total = seed + imm + mat [+ empty]

Plant cover is: 100*(imm+mat) / (nc*nr) [nc, nr: number of columns and n of rows]
Number of cells with seeds: plseed + seed, which is =< seedt =< 5*nc*nr
Empty cells = nc*nr - seed + imm + mat

Examples:

```
d1 <- disturbance(0.001)
d2 <- disturbance(0.05)
sp1 <- species("Quercus", ...)
sp2 <- species("Pinus", ..)
myland <- distribution(landcape(50,50), c("sp1", "sp2"), matrix,
c(1,0,2))
res1 <- run(100, myland, dist=c("d1", "d2"), save=c("final",
"distrec"), file= "res")
# this will produce an object called res1 plus two files: res.final and res.distrec
res2 <- run(100, myland, dist=c("d1", "d2"), save=c("final",
"distrec"), file= "res")
colmean(res1) # means by columns

rowmean(res1$imm, res1$imm)

run(100, myland, plot=T) # for visualisation; not output produced
```

run(..., plot=T)

melca – this is the friendly GUI of `run()`.

File menu:

- Save current image
- Save current plot
- Copy current image to the clipboard as metafile
- Copy current plot to the clipboard as metafile
- Save data displayed in the current plot to lass
- ....

Top bar (buttons, etc) common to plot and grid:

- Run 1 year
- Run n years; n being the number indicated in the "N.years" edit window
- Run n years without updating the grid until the end of the simulation. If the grid is visible, this would speed up the simulation (if not, then it has no effect on the speed).
- See grid/plot
- N. years: number of year to run
- step: time step of the output (in plot, grid and save)

- Edit (in grid only) permits introducing species in any cell (and choosing the age), including static cells, and removing plants from cells.

Grid:

- Edit (in grid only) permits introducing species in any cell (and choosing the age), including static cells, and removing plants from cells.

Plot:

- choosing variables to plot

Examples:

```
set.seed(123)
run(t=100,land=land1, plot=T)

set.seed(123)
res <- run(100,land1, plot=T)
```

would run 100 years, showing the plot, allowing the user to see the grid, and to pause/continue. Results are saved in the res for 100 years or until the icon stop is pressed. Further simulations are not saved in res. To save further simulations: File/Save .....

`run(0, land1, plot=T)` # for interactive menu-driven simulations

species, species.edit

Melca – set the species parameters for the Melca model by generating a list type object. `species()` generates a species variable (for each species). `species.edit()` is the same as `species()` but with a graphic interface; it also shows the shape of the dispersal function depending on the parameters chosen.

species(name, recr=no, respr=no, fuel, seedlong, maxage, matureage, lform, shadetol, sdisp, mdisp, ldisp, kdisp, limit, fecund, wind, rgrowth, col)
species.edit( )
species.edit(sp)

name – name of the species
recr – recruitment stimulated by fire: no (def), low, med, high, all
respr – post-disturbance resprouting capacity: no (def), low, med, high, all
fuel - combustibility index 0 to 9; a value or a vector of two elements, the mean and the CV
seedlong – longevity of seeds (years). A vector of two elements, the mean and the CV
maxage – maxim age (years). A vector of two elements, the mean and the CV
matureage – age at whith maturity is reached (years). A vector of two elements, the mean and the CV
lform – life form: "tree", "shrub" (def), "herb", "pinus". Pinus is (obviously) considered a tree, but with different fuel properties [XXX].
shadetol – Shade tolerance: low, med, high
sdisp – short-distance dispersal capacity: "no", "high", "med", "low" (def= med). Short-distance dispersal is a randomly uniform probability of dispersion from the cell to the first limit of the parameter *limits*.

mdisp – medium-distance dispersal capacity: "no", "high", "med", "low" (def= med). Theses parameters (high, med, low) are related to the initial dispersal probability (0.75, 0.5, 0.25) of the decay function, from the first limit to the second. The actual dispersal parameter depends also on the decay rate (kdisp). Prob. of dispersal = med * exp(-kdisp * (distance-limits[1]) / limits[2]) where distance is chosen randomly.

ldisp - long-distance dispersal capacity: "no", "high", "med", "low" (def= med). This is a uniform function (as the short distance), with dispersal probability equal to the minimum of the decay function for the medium-dispersal for the same class.

kdisp – decay rate for medium-distance dispersal (a value or a vector with mean and CV). [def= 2]. Higher kdisp (lower -kdisp) implies lower probability of dispersal for any distance (see parameter med). If kdisp= 0, then the dispersal probability is the same at any distance and equal to the med value (0, 0.25, 0.50, 0.75 for no, low, med, high, respectively).

limits – 3 values (in m) referring to the limit between short and medium dispersal, medium and long dispersal, and the maximum dispersal distance. Default values are 10, 100, and D (D= the diagonal of the landscape). Note that if limits[1]= 0 and limits[2]=limits[3], then only the decay dispersal of the medium-dispersal is applied. If uniform dispersal with distance is sought, then limits[2] and limits[3] should be the same.

fecund – qualitative measure of the species fecundity (amount of seeds produced): "no", "low", "med", "high". fecund="no" would produce no dispersal, independent of sdisp, mdisp and ldisp. It refers to the number of random distances for dispersal that are chosen (5, 10, 15). For each distance, dispersal may occur depending on the shape of the dispersal function (i.e., depending on the dispersal capacity for that distance and the kdisp).

wind – if wind= 0, dispersal is not affected by wind (wind-independent ,e.g., animal-dispersed only). A non-zero value indicates that wind is considered; see `landscape()`.

rgrowth – radial growth in meters per year (def= 0). A value or a vector with the mean and the CV. An empty random cell is chosen if radial growth reaches to the centre of the next cell (if age*rgrowth >= cell size). A new immature plant (age= 0) is allocated to the new cell.

col – a matrix with two columns (the first for mature and the second for immature) and 3 rows (with the RGB values, between 0 and 255).

sp – a species object

species.edit(sp) open up a window that permits paramenters of the species to be changed using a friendly interface. If sp is omitted, empty data (with default values) are presented.

Example:

```
sp1 <- species("spA", respr="low", fuel= 3, seedlong= 5, maxage= c(50,
    5), matureage= c(8,5), lform="woody", shadetol="med", sdisp="low",
    mdisp="low", ldisp="low", col=matrix.col(c(255,0,0), c(150,0,0)))
sp2 <- sp1
sp2$respr <- "no"
sp2$name <- "spB"
sp2$col <- "blue"
sp3 <- species.edit(sp1) # modify using a friendly window interface
sp4 <- species.edit()  # create new species
```

FATELAND – brief description

Fateland is a spatially explicit version of the FATE model (by Moore & Noble 1990) ...

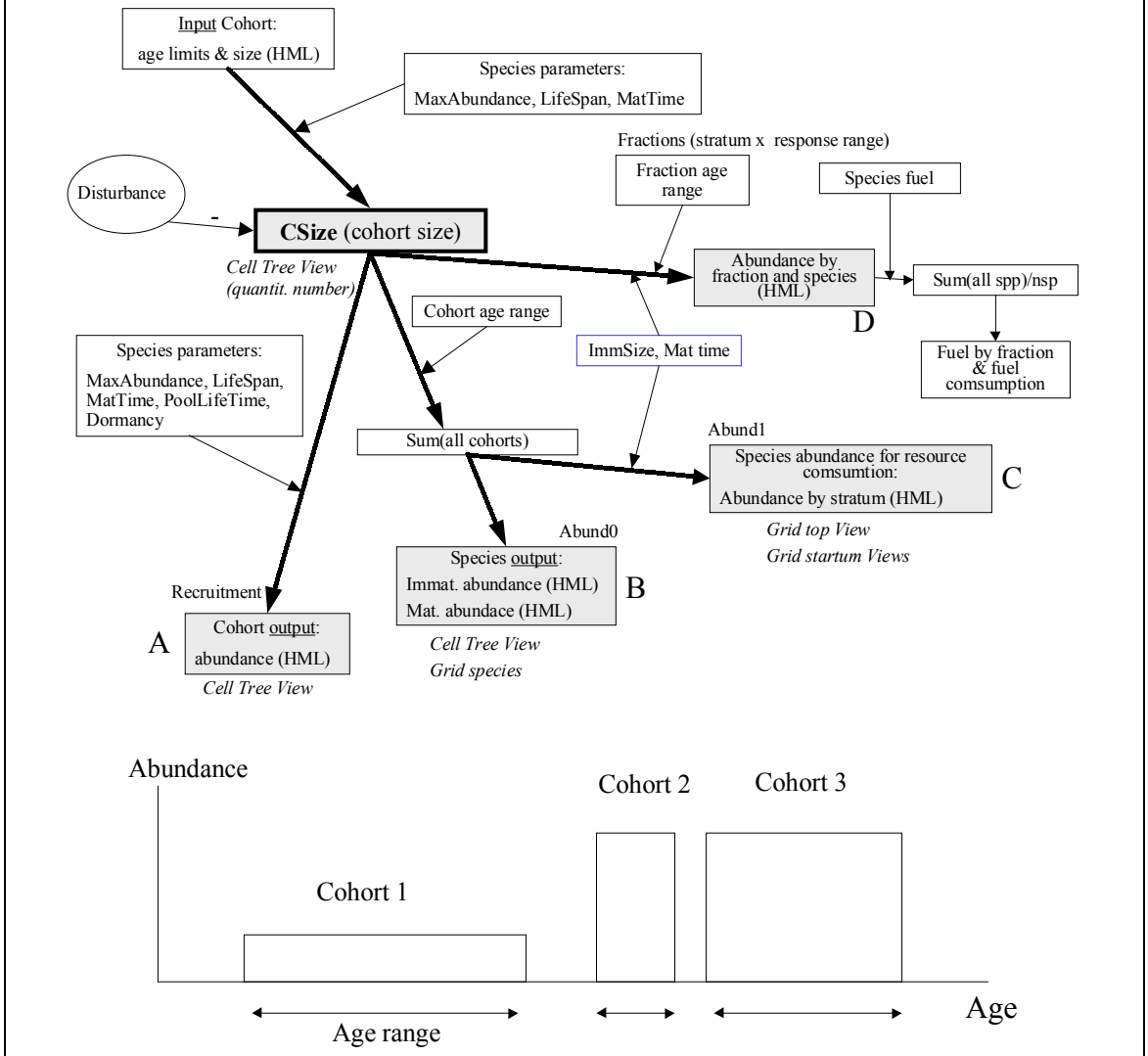
Basic characteristics:

- multiple species in each cell
- multiple response to disturbance for each species, so disturbance severity can be considered

Basic example:

```
library(fateland)
sp1<- qu #sp1 <- species.edit()
m <- pattern(matrix(0, 50, 50), 1, "rand", 0.75)
land <- landscape(50, 50)
land <- distribution(land, sp= "sp1", m, ages= c(5, 50))
res <- run(100, land)
```

The different abundant measures used in FATELAND (computed as defined in FATE).



A: only "height" (qualitative abundance) is considered, not the number of years (the range). For each cohort. In this case, size of cohort 2 = size of cohort 3
B: considering "height" and age range. For each species.
C: as B but considering that Immatures are smaller than matures (ca. biomass?). For each stratum of each species.
D: as C but for each fraction of each species (fraction= stratum x disturbance age range)
Csize is a number representing the cohort size when the MaxAge (LifeSpan) is known; for the same input abundance (LMH) and age range, Csize is larger if MaxAge is larger.

Lass functions for fateland

distribution, disturbance, landscape, species, run,

distribution, distribution.prop

fateland – modify the object generated by landscape by including the position and age of the species (plants and propagules). That is, they generate the initial condition for

running fateland. Generate a list, equal to landscape given in land=, but with two new added fields: **species** (the names of the species object), **distplants** (a matrix with the following six columns: x, y, sp, coab, ay, ao) and **distprop** (a matrix with the following five columns: x, y, sp, pab). distprop is generated by `distribution.prop`; x and y are the coordinates, sp, the code for the species (order following the field species), coab the cohort abundance (1,2,3 for low, med, high), ay and ao the age range of this cohort (min and max), and pab the propagule abundance (1,2,3). `distribution` sets plants and `distribution.prop` sets propagules, in the same landscape list.

distribution(land, sp, matrix, ages, over = "add")
distribution.prop(land, sp, matrix, over = "add")

land – object generated by landscape.
sp – the name of a species object (between ""). Species objects should be in the workspace, generated by `species()`. Only one species can be assigned (as opposed to the melca library).
matrix – a matrix with the distribution of the species and the cohort abundance. In each cell: 0: absence, and 1, 2, and 3: presence with low, medium and high abundance (of plants, in `distribution`; of propagules in `distribution.prop`). Any negative value will set a static cell (static cells are empty cells that do not change with time, e.g., outside area, non-vegetation, etc.). This matrix is often generated with `matrix()` and/or `rbinom()` and/or `pattern()`.If the *matrix* and *land* paramenters have different dimensions, the *matrix* paramenter is not redimensioned (see `pattern` for redimensioning from small matrices), and only the overlying section is used.
ages – Ay, Ao, and CV for each cohort (rows). A matrix with 2 or 4 columns in the following order: ay, ao, and optionally aycv and aocv; and as many rows as cohorts. If one cohort only, then *ages* is a vector.
over – 3 options (different when the cell is occupied):
"add" - add the cohort/seed to the cell (default)
"replace" - if occupied, it replaces the cohort/seed of the cell with the new one
"empty" - add the cohort/seed to the cell only if the cell is empty

Examples:

```
# Example 1
land <- landscape(50, 50)
mat <- matrix(3, 50, 50)
land3 <- distribution(land3, sp= "sp1", mat, ages= matrix.row(c(7, 8,
    5, 5), c(5,5,5,5), c(0,0,0,0)) )
land3
str(land3)
mat <- matrix(0, 50, 50)
mat[5,10] <- 2
land3 <- distribution(land3, sp= "sp2", mat, ages= c(0, 15, 10, 10))
mat <- matrix(0, 50, 50)
mat[5,40] <- 2
land3 <- distribution(land3, sp= "sp3", mat, ages= c(0, 15, 10, 10))
```

```
# Example 2
fireresp <- species.disturbance(...)
quercus <- species(....)
```

```
pinus <- species(....)
land0 <- landscape(10, 10)

m1 <- matrix(rbinom(100, 3, prob=0.5), 10, 10) # all kinds of abundance
m2 <- matrix(rbinom(100, 3, prob=0.8), 10, 10) # all kinds of abundance
    biased towards high
m3 <- matrix(rbinom(100, 3, prob=0.1), 10, 10) # all kinds of abundance
    biased towards low/absence
m4 <- matrix(rbinom(100, 2, prob=0.5), 10, 10) # 0,1 or 2 evenly distributed

# other ways to generate distribution/abundances using pattern (they are not the same
    as the previous ones!, they are just more examples):
m1 <- pattern(matrix(0, 10, 10), 1, "rand", par= 0.3) # ca. 30% of 1 (low
    abundance) randomly distributed
m2 <- pattern(matrix(0, 10, 10), 2, "rand", par= 0.3)
m3 <- pattern(matrix(0, 10, 10), 3, "rand", par= 0.1)
m4 <- pattern(matrix(0, 10, 10), 1, "rand", par= 0.5)

land1 <- distribution(land0, sp= "quercus", matrix=m1, ages= c(10,
    15)) # set one cohort only
land1 <- distribution(land1, sp= "quercus", matrix=m2, ages= c(50,
    60)) # set a 2nd cohort in other cells (or in the same)
land1 <- distribution(land1, sp= "pinus", matrix=m4, ages= c(1, 5)) #
    set one cohort for pinus in the cells where there are quercus and pinus
land1 <- distribution(land1, sp= "pinus", matrix=m3, ages= c(100,
    120))# set some old pinus in some places in the landscape
land1 <- distribution.prop(land1, sp= "pinus", matrix=matrix(1, 10,
    10)) # set propagules of pinus in all cells (with low abundance)

# for an empty landscape, distribution is not needed, e.g.
land <- landscape(50, 50)
land$species <- c("quercus", "pinus")
# the same could be done by including an empty species: land <-
    distribution(land, sp= "quercus", matrix(0, 5, 5), ages= c(0, 0) )
run(0, land, grid=T) # from the grid, species can also be included using the mouse
    (edit landscape)
```

disturbance

fateland – Generate an object (list) with the disturbance characteristics for run the FATELAND model. `disturbance.edit()` is a friendly window interface version for `disturbance()`. [not yet available]
This function is similar to the disturbance function in Brolla, and the list generated can be used in any of the two models (the only difference is that *stratum* is used in Fateland and not in Brolla).

disturbance(tpar, ttype= "rand", first= 1.00, sn= 1.00, stype= "rand", sscope= null, loc= "", around= "TRUE", ptype= "fuel", ppar= null, pscope= null, stratum= "").

distobj – a disturbance object generated with `disturbance()`.

-- time parameters:
ttype – disturbance frequency type:
"rand": randomly
"syst": systematic
"weib": weibull distribution
"speci": disturbance in specific years
tpar – disturbance frequency parameter.
if "rand", then the disturbance probability (0-1)
if "syst", the disturbance interval or a vector with the mean disturbance interval and the CV
if "weib", then average disturbance interval and the k coefficient for the Weibull distribution. if k= 1, then it is a negative exponential distribution (def: 2.3)
if "speci", then a vector with the disturbance years.
first – first (and last) year in which the parameters function `disturbance()` is applied. A single value (integer) or a vector of 2 values (integers), first and last. It is ignored if tpar = "speci". Default is first = 1, which means that the disturbance regime starts at the beginning of the simulation and finishing at the end of the simulation.

-- space parameters (e.g., location of ignitions):
sn – number of disturbance events (an integer) or a vector with the means and the coefficient of variation (%) of the number of disturbance events in each disturbance year (def.: sn=1).
stype – how the disturbance initial cells (e.g., fire ignition cells) are chosen:
"rand": randomly (default)
"syst": systematically; if stype = "syst", then `loc` is needed (e.g., fire ignition cells), and sn is not considered.
"fuel": fuel random. Fuel-driven fire.
loc – a matrix of 2 columns (x and y coordinates) and as many rows as the number of initial disturbance points. sn is not considered (the number of initial points is the number of rows).
sscope – spatial scope: Area in which disturbance may start, defined by a matrix of 0 (not possible starting) and 1 or any non-zero (possible starting; any number different from 0 will behave as a 1). If not included (default), then the disturbance can start in any place of the landscape. This is only applicable for stype= "rand". To be precise, then around should be FALSE, otherwise disturbance may start outside of the sscope area.
around – in case the initial cells are not burnable (empty or seeds only), whether the fire should start in the closest cell (def: around = yes). Available for `stype= "rand"` or `"syst"`. Note that if a coefficient of variation is given in `sn` and `around= FALSE`, then the annual number of disturbances is quite unpredictable, but most probably lower than the mean value of `sn`. Note also that if `around` is used together with `sscope` then, disturbance may start outside of the area with sscope = 1.

-- propagation parameters:
ptype – type of propagation:
"fuel" – probabilistic fuel-driven (simple fire disturbance). (default)
"prob" - based on a constant probability (0-1)
"radius" - disturbed everything within a square of radius (1/2 side) of ppar

"all" - all landscape is disturbed. Space parameters (`sn, stype, loc, around`) are ignored.

"hier" – hierarchically structured pattern. Space parameters (`sn, stype, loc, around`) are ignored.

"matrix" – disturbed pattern as in the matrix provided in `ppar`. Space parameters (`sn, stype, loc, around`) are ignored.

ppar – propagation parameter:

if ptype = "prob", then probability (0-1). Optionally, a second parameter (`nint`, typically between 1 and 8), meaning the number of interactions in each cell can be provided (default= 4); in that case, ppar would be a vector of 2 parameters. This second parameter refers to the number of times that the probability for propagation is tested in each cell when "burning", and can be related to fire duration in the cell. If the number is low (few interactions), the propagation is lower (less area burned). There is an interaction between the prob and the nint (see Figure/Comments below).

if ptype = "radius", then length of the radius in number of cells

if ptype = "all", then ppar is ignored.

if ptype = "hier", then it is a matrix with x and y levels and probabilities (in this order: first, second and third column) for hierarchically structured random landscapes.

if ptype = "matrix", then it is a matrix (lass object) or a vector with the names of the different matrices. Matrices are 0 for the non-perturbed and any other number for disturbed. If more than one matrix is included, but the number of disturbances is higher than the number of matrices, then the matrices are recycled, that is, it starts again for the first matrix as many times as necessary.

if ptype = "fuel", then ppar is the maxim proportion of landscape able to burn (by default = 0%!!, thus it is necessary to set this parameter is fires are desidered. An appropiate value could be 100%). If 2 values are included, the first in the mean and the second de CV (default ppar = c(100, 0)). This can be used to limit the fire size.

pscope – Area in which disturbance may be propagated (by default all landscape). This is a matrix of 0 (no propagation, e.g. unburneable area) and 1 or non-zero (propagation is possible; any number different from 0 will behave as a 1). If not included, then disturbance can be propagated to all landscape, depending on the other propagation options.

-- Severity parameters

stratum - Strata affected by the disturbance; a number or a vector with integers < 5. Def= seq(1, 5). Note that if `matureage` has random variability (CV) and disturbance affects only a specific stratum, then after disturbance some plants in the lower stratum could be mature and some plants in the upper stratum could be immature. This effect will last only for the first post-disturbance year.

Examples:

```
rfirehigh <- disturbance(0.05)
rfirelow <- disturbance(0.01)
# similar, but different fire regimes are:
rfirehigh2 <- disturbance(tpar= c(20, 10), "syst")
rfirelow2 <- disturbance(tpar= c(100, 10), "syst")

xy <- matrix.col(x= c(1, 2, 3, 4), y= c(10, 20, 30, 40))
```

```
rfire2 <- disturbance(0.05,"rand", stype= "syst", loc= xy)

fsys20surface <- disturbance(20, "sys", ptype = "all", stratum= 1) # all
```
cells disturbed every 20 yrs, but affecting only the lower strata

```
# creating disturbance patterns
hmat <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(1,0.5,0.9) )
distmat <- pattern(matrix(0, 50, 50), 1, "hier", hmat1)
mydist1 <- disturbance(0.05, "rand", ptype= "matrix", ppar= distmat)
mydist2 <- disturbance(0.05, "rand", ptype= "hier", ppar= hmat)
```

both mydist1 and mydist2 are hierarchically random disturbance patterns; however the differences are:
mydist1: in every disturbance year the cells disturbed are the same, and are given by the matrix `distmat`
mydist2: `hmat` is the matrix providing the parameters for the hierarchical random disturbance pattern, and so the exact cells disturbed are different each time

```
# rotation example (option "matrix")
m1 <- matrix(1, 5, 5)
m2 <- matrix( rep(c(rep(0, 5), rep(1, 5)), 5), 10, 5 )
m3 <- matrix(c(rep(0, 50), rep(c(rep(1, 5), rep(0, 5)), 5)), 10, 10 )
m4 <- matrix(c(rep(0, 50), rep(c(rep(0, 5), rep(1, 5)), 5)), 10, 10 )

# or the same:  (option matrix not yet working)
# m <- matrix(0, 10, 10)
#m1<- pattern(m, "matrix", matrix.row(c(1,0), c(0,0))
#m2<- pattern(m, "matrix", matrix.row(c(0,0), c(1,0))
#m3<- pattern(m, "matrix", matrix.row(c(0,0), c(0,1))
#m4<- pattern(m, "matrix", matrix.row(c(0,1), c(0,0))
```

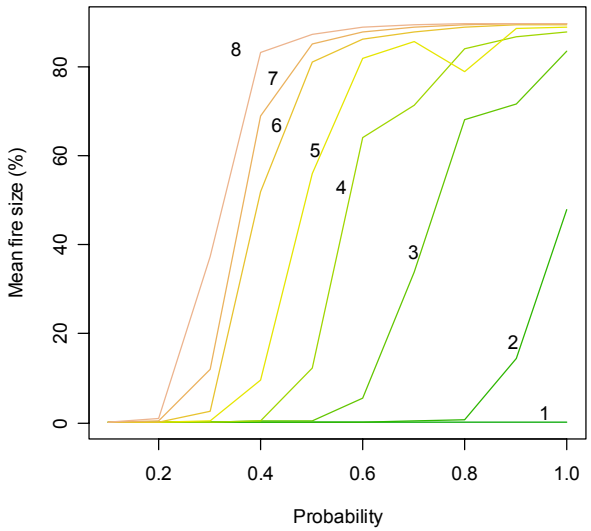
rotation <- disturbance(ttype= "speci", tpar= seq(0, 20, by=2), ptype="matrix", ppar= c("m1","m2","m3","m4")) # rotation, disturb a different quadrant every 2 years.

Comments:
The effect of ppar (probability and number of interactions) for ptype="prob".
using R:

```
qq <- function(p, n=8) {
 q <- 0;  for (i in 1:n) q <- q + p*(1-p)^i
 q
}
p <- seq(0, 1, by= 0.05)
plot(p, qq(p), xlim=c(0,1), ylim=c(0,1), type="n")
for (i in 1:8) lines(p, qq(p, i), col= i)
```

```
Examples

# lass example
# define a landscape (e.g., 'land') with species(), landscape(), and
   distribution().
# Then:
a <- matrix(0, 8, 10)
source("fire.prob.lass", 10) # file below
a # to see the results (see Figure below).


---------- file:fire.prob.lass ---------------
f0 <- disturbance(5, "syst", ptype="prob", ppar= %/10)

f0$ppar<-c(%/10, 1)
```

```
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,1]<-mean(r$disturb)

f0$ppar<-c(%/10, 2)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,2]<-mean(r$disturb)

f0$ppar<-c(%/10, 3)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,3]<-mean(r$disturb)

f0$ppar<-c(%/10, 4)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,4]<-mean(r$disturb)

f0$ppar<-c(%/10, 5)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,5]<-mean(r$disturb)

f0$ppar<-c(%/10, 6)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,6]<-mean(r$disturb)

f0$ppar<-c(%/10, 7)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,7]<-mean(r$disturb)

f0$ppar<-c(%/10, 8)
set.seed(1)
r<-run(50, land, tstep=50, dist=f0)
a[% ,8]<-mean(r$disturb)
--------------endfile--------------
```



Effect of disturbance propagation probability and number of interactions (1 to 8) in the mean disturbance size (as percent of the total landscape) for a given landscape (computed with the file `fire.prob.lass` in the example).

landscape

fateland – generates a object with the characteristics of the landscape

landscape(nr, nc, psize = 10, stratumh= -1, wd = NILL, wi = 0, topo)

nr – number of rows of the gridded landscape
nc – number of columns of the gridded landscape
psize – pixel size (real value); length of the side of the squared pixel (def = 10)
stratumh - stratum heights (in m), an optional vector with the heights of each stratum, from bottom to top. If not included (then a negative value is assigned), all stratum are considered to be the same height (as the original FATE). Currently, it is only used for computing the biomass (fuel) available for burning during a fire, if disturbance(ptype="fuel"). E.g. $stratumh <- c(0.5, 2, 5) means that the landscape has 3 strata, the first up to 0.5 m, the second up to 2 m and the third up to 5 m.
wd – direction of the dominant wind (def = NILL, no wind): "n", "ne", "e", "se", "s", "sw", "w", "nw"
wi – intensity of the wind ("none", "low", "med", "high").
topo – name of the terrain (topography) matrix objects (topo = "mytopo")

Example:

```
qu <- species.edit()        # to be filled from a menu
pi <- species.edit()
myland <- landscape(50, 50)   # 50 x 50
myland$species <- c("qu", "pi") #  include species. See also distribution
run(100, myland, grid=T)    # species can be located in the grid using the mouse
```

run, run.load

fateland – Execute the simulation for t years. `run` can be used with a graphic user interface (GUI) by setting the grid or plot options to TRUE (in that case, t is not needed). Run can be stopped/aborted using Ctl-Q (when running in CUI), etc.... run.load loads a previously saved binary file (see save2 option) as initial conditions for the run, and inserts the necessary species, landscape and disturbance lists.

run(t, land, sp, dist, tstep=1, plot=F, grid=F, save, save2, savet, savef=F)
run.load(t, file="", tstep=1, plot=F, grid=F, save, save2, savet, savef=F)

t – number of years
land – landscape objects generated by `landscape()`
sp – a vector with the name of the species object to be included. Species objects are generated by `species()`. Species list is already included in the landscape object (typically included using `distribution()`), and so this option is only needed if only a subset of species are desired in the run.
dist – the name of a disturbance object or a vector with the name of several disturbance objects. Disturbance objects are generated by `disturbance()`. If not included, no disturbance.

tstep - time step for displaying and saving the results (default, tstep=1, annual time step). If disturbance is applied, then the disturbance year is also displayed. That is, if only disturbance years are desired, then use a `tstep >= t`.

grid, plot – FALSE (default)/TRUE for the use of a graphic interface. If these options are set to T, then they cannot be used in a source file.

save – a save option or a vector of save options. Save options in the assigned results object in run:

"ctotal": counts total, number of cells occupied each year (incl mat, imm, seeds, ..., disturbances) (Default)

"cspp": counts by species (as above for each species)

"cdist": count the number of cells for each disturbance type.

save2 – additional saved objects that can be generated during the run. It corresponds to one of the options below, or to a vector with several options. The name of the resulting object will be the one given to assign the run results (maximum 8 characters), followed by "_" the abreviated name of option (topv, str1, str2, .., rich, drec) and the simulation year in which the save was performed (e.g. res_topv_50). In the case of the species matrices, the name of the species (in the field $name of the species object) is used instead of the option (res_Quercus_50). Options are:

"spp": matrix (map) for each spp with values of 1 to 6 for Immature low, medium, high and mature, low, medium and high; 0 for absences; and values –1, -2, -3 and -4 for seeds only (low-dormant, high-dormant, low-active and high-active, respectively). If both active and dorment are present, only the active are indicated.

"topview": a matrix with the dominant species in the top stratum of each cell. Abundance of each species and seeds are not considered. If it is a fire year, then burned cells are indicated with a negative value.

"stratum1", "stratum2", ..., "stratum5": a matrix with the dominant species in each cell of the stratum. If it is a fire year, then burned cells are indicated with a negative value.

"richness": a matrix with the number of species in each cell.

"distrec": disturbance recurrence. A matrix (map) with the number of disturbances in each cell.

"dist": disturbance pattern (0/1) (it saves as many maps/matrices as disturbances). Matrices are stored with the name assigned to the run results followed by "_dist_X" where X is the simulation year (in this case, savet is not considered).

"xyall": a matrix with the columns: row, column, species, number of cohorts, max and min age, abundance of Immature and Mature plants, and abundance in the active and dormant seedpool. Note that this can be a very large matrix.

"binaryland": save the whole landscape in binary form to a file. It can be reloaded by run.load() . The name of the file will be the name assigned to the run results followed by "_binaryland_X" where X is the simulation year. It can also be saved from the GUI (menus).

savet – (savetime) a value or vector indicating the simulation year in which save2 is performed. A -1 (default) indicates to save at the end of the simulation. Eg: `$savet <- seq(0, 100, by= 10)` would save data every 10 years. `$savet <- c(0, 50, -1)` would save data before starting the simulation, at year 50, and at the end. The vector should be sorted, except for –1, which should be the last value (otherwise, the save process will stop after -1).

savef – (saveToFile) By default (savef=F), the save2 options are stored in the lass environment. By setting savef=T, the information on save2 is stored in ascii files (comma-separated) in the current working directory; this files can be read by many programs; lass can read them using `read("filename", sep=",")`. savef do not apply to "binaryland" which is always saved as a binary formatted file.

file – Name (and path, if it is not in the working directory) of the binary file saved using save2= "binaryland" (or from GUI menus). It is used for running the model from the initial conditions of a previous run. This file contains all the information for running FATELAND, including landscape, species and disturbance lists.

-- the main bar options

-- the grid: run(...., grid=T, ...)

-- the plot: run(...., plot=T, ...)

Examples:

```
run(land=myland, grid=T) # opens up the GUI with all information in myland
#
res <- run(100, myland, dist= myfire, save= "cssp", save2=
    c("topview", "distrec"), savet= c(1, 50, 100))

# tstep: one way to see only the results of the years with disturbance
run(100, myland, dist=fire10, tstep=100)
```

species, species.edit

fateland – set the species parameters for the fateland model by generating a list type object. `species()` generates a species variable (for each species). `species.edit()` is the same as `species()` but with a graphic interface; it also shows the shape of the dispersal function depending on the parameters chosen. `species.edit()` includes also `species.disturbance()`.

species(name, maxab, matureage, agemax, size, stratum, proppool, germ, survival, distresp, col, sdisp, mdisp, ldisp, kdisp, limits, fecund, fecundfire, fecundmast, wind, rgrowth, fuel, canopy=100)

species.edit( )
species.edit(sp)

name – species name (a string). Blank spaces and other reserved characters (-,+,/,=,*,:,$,<,>, ...) are ignored; underscore '_' is accepted.

maxab - Maximum abundance (1,2,3 for low, medium, high). Default= 3 (high).

matureage - age at which maturity is reached (years). A value or a vector with two elements, the mean and the CV. The mean matureage value is used to change stratum (from lower to higher). If CV>0, then there will not be an exact relation between stratum and maturation state; that is, some plants in the upper stratum could be immature for some time, and some plants in the lower stratum could mature before reaching the upper stratum.

agemax - maximum age (years). A vector of two elements, the mean and the CV.

size - size (or proportion of resource consumption) of immature relative to mature plants (0, 1, 2, 3, 4 for none, few, half, most, all). Def; size= 1 (few).

stratum – a vector of 2 values, the first for immatures and the second for matures. If only one value is provided, it is assumed that both immature and mature are in the same stratum. Stratum correspond to the vertical layer that reaches the immature and the mature plants. Default= `c(1, 2)`. Mature plants are in one stratum only (if matureage CV=0) in the upper stratum; but immatures can occupy several strata if there is no continuity between the two values provided. In such cases, the transition is assumed lineal in time (it can be improved xxx). E.g.: if matureage = 10 and stratum = c(1, 3), then, plants will stay in stratum 1 for 5 years, in stratum 2 for 5, years and at the age of 10 they will mature and pass to stratum 3.

Note 1: In case of no continuity in strata between immatures and matures, sizeim (the relative size of immatures) is also estimated as a lineal increase.

Note 2: The assumptions for intermediate strata are different from the original FATE, in which no intermediate states were possible (in our example, the original FATE plants would pass from strata 1 to 3 without being in 2).

Note 3: The stratum values should be consistent with the number of strata considered in the `landscape()` by the parameter `stratumh`.

proppool – a vector of 1 to 4 elements which are the (mean) life span (in years) of the active and the dormant propagule pool and the CV for active and dormant, respectively. E.g., proppool= c(5, 50, 5, 5) means that the life span of the active propagule pool is 5 year and the life span of the dormant propagule pool is 50 years, with a coefficient of variation of 5% for both. proppool= 5 would mean that the species has no dormancy and CV is not provided. In this example CV can be provided by: proppool= c(5, 0, 1) = c(5, 0, 1, 0). LifeSpan of propagules can be 0, which means that only during the first year can they germinate; thus c(0,5,0,0) is not the same as 5.

germ - germination rate (0, 1, 2, 3, or 4 for none, low, medium, high, all). A vector with 3 elements, for germination at low, medium and high resource levels. E.g., germ = c(0, 1, 4)

survival – matrix of 9 elements (0/1 for yes/no): in rows, survival of germinants (seedlings), immatures, and matures; in columns, survival for low, medium and high resource levels for each row. Default = matrix(rep(1, 9), 3, 3). E.g., matrix.col(low=c(0,0,0), med= c(0,0,1), high= c(1,1,1)); the same can be generated by rows, i.e., matrix.row(ger= c(0,0,1), imm=c(0,0,1), mat=c(0,1,1)).

distresp – it can be:

1) the name of the disturbance response objects generated by `species.disturbance()`, or

2) a matrix (or a vector with 2 elements) with the names of the disturbance generated by disturbance( ) and the corresponding names of the disturbance responses generated by `species.disturbance()`.

If omitted, then species does not respond to disturbance (no resprouting and no survival of propagules).

E.g., in c("fire1", "resp1"), the first is the disturbance and the second the response; in `matrix.col(c("fire1", "grazing"), c("resp1", "resp2"))` which is the same as `matrix.row(c("fire1", resp1), c("grazing", resp2))`, the first column are disturbances and the second column are the corresponding responses.

col – basic colour for the species in the grid expressed as RGB (vector of 3 values between 0 and 255). Final colours will change (a gradient of this basic colour) for

immatures and for abundance (low, med, high) of matures. Currently it needs to be included as a column, e.g., t(c(255,0,0)), (i.e., red.)

sdisp – short-distance dispersal capacity: "no", "high", "med", "low" (def= med). Short-distance dispersal is a randomly uniform probability of dispersion from the cell to the first limit of the parameter *limits*. Values are 0.9, 0.6 and 0.3 (and currently they cannot be modified).

mdisp – medium-distance dispersal capacity: "no", "high", "med", "low" (def= med). These parameters (high, med, low) are related to the initial dispersal probability (`sdisp`) of the decay function, from the first limit to the second. The actual dispersal parameter depends also on the decay rate (`kdisp`). Prob. of dispersal = med * exp(-kdisp * (distance-limits[1]) / limits[2]) where distance is chosen randomly and med is 0, 0.3, 0.6, 0.9 for "no", "high", "med", "low" mdisp.

ldisp - long-distance dispersal capacity: "no", "high", "med", "low" (def= med). This is a uniform function (as in the short distance one), with a dispersal probability equal to the minimum of the decay function for medium-dispersal for the same class.

kdisp – decay rate for medium-distance dispersal. [def= 2]. Higher kdisp (lower -kdisp) implies lower probability of dispersal for any distance (see parameter med). If kdisp= 0, then the dispersal probability is the same at any distance and equal to the med value (0, 03, 0.6, 0.9 for no, low, med, high, respectively). It can be a vector with two elements, the mean and the CV.

limits – 2 or 3 values (in m) referring to the limit between short and medium dispersal, medium and long dispersal, and the maximum dispersal distance. Default values are 10, 100, and D (D= the diagonal of the landscape). Note that if limits[1]= 0 and limits[2]=limits[3] then only the decay dispersal of the medium-dispersal is applied. If uniform dispersal with distance is sought, then limits[2] and limits[3] should be the same. If limits[3] is not included, then D is assumed.

fecund – Number of distances randomly chosen each year; that is, the maximum viable "seeds" dispersed. It is a measure of the species fecundity (amount of seeds produced). fecund= 0 would produce no dispersal, independent of sdisp, mdisp and ldisp.

fecundfire – for serotiny species, the additional fecundity after fire. One or two values (mean and coeficient of variation).

fecundmast – XXXX

wind – if wind= 0, dispersal is not affected by the wind (wind-independent e.g., animal-dispersed only), otherwise wind has effects; See `landscape()`.

rgrowth – radial growth in meters per year (def= 0). An empty random cell is chosen if radial growth reaches to the centre of the next cell (if age*rgrowth >= cell size). The new cell is assigned a new immature plant (age= 0). It can be a vector with two elements, the mean and the CV.

fuel – (optional) combustibility index as %; default = 50%

canopy – Proportion of the plant height with canopy (with burnable fuel). Used for estimating the probability that the fire will spread vertically when `disturbance(ptype="fuel")`. By defaults it is 100% (as many shrubs).

species.disturbance

fateland – generate a list with the disturbance response parameters. The name of the list is the input for `distresp[ ,2]` in the function `species()`. It is also included in the GUI species menu, i.e., `species.edit()`.

species.disturbance(seedbroken=0, propkill=4, agelim=0, killresp=c(4,0), respage=1)

seedbroken - fraction of seeds with dormancy broken by disturbance (0, 1, 2, 3, 4, for no, low, medium, high)

propkill – proportion of propagules (seeds and vegetative reproductive structures) killed by disturbance (no, few, half, most, all). Default: 4 (all).

agelim – age limits for different responses; a vector with different elements (integers, years) that are the upper age limit. Example, `$agelim <- c(3, 40)`, then there are 3 age classes, <3, 3-40, and >40 years. If agelim is not included or agelim=0, then only one response (for any age) is assumed.

killresp – proportion of individuals killed and resprouting:

A. if only one response is used (same response for any age), then killresp is a vector of two elements: proportion of individuals killed (0, 1, 2, 3, 4, for none, few, half, most, all) and proportion of individuals resprouting (0, 1, 2, 3, 4, for none, few, half, most, all). The remaining are unaffected; note that: all = killed + resprouting + unaffected; e.g., all (4) = none (0) + most (3) + few (1) = none (0) + none (0) + all (4) etc.

B. if there are several responses, depending on the age (see agelim), then a matrix with a row for each age class and 2 columns: proportion of individuals killed (0, 1, 2, 3, 4, for none, few, half, most, all) and proportion of individuals resprouting (0, 1, 2, 3, 4, for none, few, half, most, all).

Default: all killed for any age, c(4, 0)

resprage – Functional age after resprouting. A vector with one element for each age interval (see agelim). If omitted and it resprouts, then it is assumed to be = 1 (i.e., starting to resprout as a 1 year-old seedling).

Examples:

```
sp1 <- species(...)
sp2 <- species(...)
```

plants resprouting vigorously at any age, and after any disturbance

```
resp <- species.disturbance(killresp= c(0,4), respage= 10) #
sp1$distresp <- "resp"
```

3 disturbances: 2 fire frequencies and a systematic matrix disturbed (e.g. grazing)

```
firehighr <- disturbance(, 0.05) # high fire recurrence
firelowr <- disturbance(, 0.01) # low fire recurrence
grazing <- disturbance( "syst", 25, ptype= "matrix", ppar= matrix(1,
    10, 10) )
```

2 disturbance responses (high and low severity)

all indiv. killed (default), i.e., no resprouting; half seed killed (and half survived):

```
highsev <- species.disturbance(propkill=2)
    # resprout, except few young and old indiv:
lowsever <- species.disturbance(c(10, 50), propkill="few", killresp=
    matrix.col(kill= c(1,0,1), resp= c(2, 3, 4) ), respage=c(5, 5, 5) )
```

link disturbance and disturbance response for a given species. For sp1: high fire recurrence with low severity & low fire recurrence with high severity; and grazing effect as a low severity fire. For sp2: all fires and grazing kill all individuals.

```
sp1$distresp <- matrix.row(c("firehighr", "highsev"), c("firelowr",
    "lowsever"), c("grazing", "lowsever") )

sp2$distresp <- sp1$distresp

land1 <- distribution(landscape(100, 100), sp = sp1, .....)
land1 <- distribution(land1, sp = sp2, .....)
str(land1)
run( t= 100, land=land1, dist=c("firehighr", "firelowr", "grazing") )
```

BROLLA – brief description

Here we present BROLLA version 2, which is a spatially explicit version of the previous BROLLA model (Pausas 1999). When referring to one of the specific versions or when comparing versions, we will call them BROLLA1 and BROLLA2; otherwise, BROLLA refers to the version implemented in LASS (i.e., the latest version, 2). BROLLA1 is a non-spatial individual-based gap model. BROLLA2 is like including BROLLA1 in a grid system, with the necessary modifications in dispersal and the fire modules for spreading into the space.

Lass functions for Brolla

distribution, disturbance, landscape, species, run,

(not yet finished)

distribution. distribution.prop

brolla – modify the object generated by landscape by including the position and size of the different plants of each species in each cell. That is, they generate the initial condition for running BROLLA. To include several plants in a cell, distribution need to be executed several times (see examples). They generates a list, equal to the landscape given in land=, but with two or three additional fields:

species - the names of the species object
distplants - a matrix with the following four columns: x, y, sp, diam
distprop - a matrix with the following four columns: x, y, sp, pab; generated by `distribution.prop`

x and y are the coordinates, sp, the code for the species (order following the field species), diam are the diameter (in cm) and pab propagule abundance (between 0 and 1). `distribution` sets plants and `distribution.prop` sets propagules in the same landscape list.

distribution(land, sp, matrix, over = "add")
distribution.prop(land, sp, matrix, over = "add")

land – object generated by landscape.
sp – the name of species objects (between " "). Species objects should be in the workspace, generated by `species()`. Only one species can be assigned (as opposed to the melca model).
matrix – a matrix with the distribution of the species and the diameter. In each cell: 0: absence, and any other number indicate the position, and value is the diameter. If the value is negative, a static cell is included (static cells are empty cells that do not change with time, e.g., outside area, non-vegetation, etc.). This matrix is often generated with the function `matrix()` and/or `rnorm()` and/or `pattern()`.If the *matrix* and *land* paramenters have different dimensions, the *matrix* paramenter is

not redimensioned (see `pattern` for redimensioning from small matrices), and only the overlying section is used.

over – 3 options (different when the cell is occupied):

"add" - add plants/seeds to the cell

"replace" - if occupied, it replaces the plants/seeds in the cell with the new one

"empty" - add plants/seeds to the cell only if the cell is empty

Examples

```
land <- landscape(50, 50)
mat <- matrix(abs(rnorm(250, 20, 2)), 50, 50) # diameters meam=20 SD=2
mat <- pattern(mat, 0, "rand", 0.3)      # 30% of the cells empty
land <- distribution(land, sp= "sp1", mat)

mat2 <- matrix(0, 50, 50)
mat2[,1] <- 50           # add big trees in all cells of the first column
land <- distribution(land, sp= "sp1", mat2)
run(, land)

# Another example, with the help of a user defined function:
pi <- species.edit()
qu <- species.edit()
ci <- species.edit()
er <- species.edit()

land <- landscape(100, 100)

dmat <- function(md) {
     # diameters meam=md CV=20%; 50% of the cells empty
  m <- matrix(abs(rnorm(10000, md, 20*md/100)), 100, 100)
  pattern(m, 0, "rand", 0.5)
}

land <- distribution(land, sp= "pi", dmat(20))
land <- distribution(land, sp= "qu", dmat(20))
land <- distribution(land, sp= "ci", dmat( 2))
land <- distribution(land, sp= "er", dmat( 4))

table(land$distplants[,3])              # n of plants of each species
table(land$distplants[,4], range=10)    # diameter freq. distribution
plot(land$distplants[,3:4], type="p")   # species vs diameters

run(, land)

# An example including the data directly to the $distplants
land2 <- landscape(100, 100)
land2$species <- c("pi", "qu", "ci", "er")
 x <- c(10,10,11,11, 80,80,81,81, 10,10,11,11, 80,80,81,81)
 y <- c(10,11,10,11, 10,11,10,11, 80,81,80,81, 80,81,80,81)
 s <- rep(c(1, 2, 3, 4), rep(4,4))
           # species 1, 2, 3, 4, refer to the order in land2$species
 d <- rep(c(10,20,1, 2), rep(4,4))
land2$distplants <- matrix.col(x=x, y=y, sp=s, diam=d )
run(, land2)
```

disturbance

Brolla – Generate an object (list) with the disturbance characteristics for run the BROLLA model. `disturbance.edit()` is a friendly window interface version for `disturbance()`. [not yet available]
This function is similar to the disturbance function in Fateland, and the list generated can be used in any of the two models (the only difference is that *stratum* is used in Fateland and not in Brolla).

disturbance (tpar, ttype= "rand", first= 1.00, sn= 1.00, stype= "rand", sscope= null, loc= "", around= "TRUE", ptype= "fuel", ppar= null, pscope= null).

distobj – a disturbance object generated with `disturbance()`.

-- time parameters:
ttype – disturbance frequency type:
"rand": randomly
"syst": systematic
"weib": weibull distribution
"speci": disturbance in specific years
tpar – disturbance frequency parameter.
if "rand", then the disturbance probability (0-1)
if "syst", the disturbance interval or a vector with the mean disturbance interval and the CV
if "weib", then average disturbance interval and the k coefficient for the Weibull distribution. if k= 1, then it is a negative exponential distribution (def: 2.3)
if "speci", then a vector with the disturbance years.
first – first (and last) year in which the parameters function `disturbance()` is applied. A single value (integer) or a vector of 2 values (integers), first and last. It is ignored if tpar = "speci". Default is first = 1, which means that the disturbance regime starts at the beginning of the simulation and finishing at the end of the simulation.

-- space parameters (e.g., location of ignitions):
sn – number of disturbance events (an integer) or a vector with the means and the coefficient of variation (%) of the number of disturbance events in each disturbance year (def.: sn=1).
stype – how the disturbance initial cells (e.g., fire ignition cells) are chosen:
"rand": randomly (default)
"syst": systematically; if stype = "syst", then `loc` is needed (e.g., fire ignition cells), and sn is not considered.
"fuel": fuel random. Fuel-driven fire.
loc – a matrix of 2 columns (x and y coordinates) and as many rows as the number of initial disturbance points. sn is not considered (the number of initial points is the number of rows).
sscope – spatial scope: Area in which disturbance may start, defined by a matrix of 0 (not possible starting) and 1 or any non-zero (possible starting; any number different from 0 will behave as a 1). If not included (default), then the disturbance can start in any place of the landscape. This is only applicable for stype= "rand". To be precise, then around should be FALSE, otherwise disturbance may start outside of the sscope area.

around – in case the initial cells are not burnable (empty or seeds only), whether the fire should start in the closest cell (def: around = yes). Available for `stype= "rand"` or `"syst"`. Note that if a coefficient of variation is given in `sn` and `around= FALSE`, then the annual number of disturbances is quite unpredictable, but most probably lower than the mean value of `sn`. Note also that if `around` is used together with `sscope` then, disturbance may start outside of the area with sscope = 1.

-- propagation parameters:

ptype – type of propagation:

"fuel" – probabilistic fuel-driven (simple fire disturbance). (default)

"prob" - based on a constant probability (0-1)

"radius" - disturbed everything within a square of radius (1/2 side) of ppar

"all" - all landscape is disturbed. Space parameters (`sn, stype, loc, around`) are ignored.

"hier" – hierarchically structured pattern. Space parameters (`sn, stype, loc, around`) are ignored.

"matrix" – disturbed pattern as in the matrix provided in `ppar`. Space parameters (`sn, stype, loc, around`) are ignored.

ppar – propagation parameter:

if ptype = "prob", then probability (0-1). Optionally, a second parameter (`nint`, typically between 1 and 8), meaning the number of interactions in each cell can be provided (default= 4); in that case, ppar would be a vector of 2 parameters. This second parameter refers to the number of times that the probability for propagation is tested in each cell when "burning", and can be related to fire duration in the cell. If the number is low (few interactions), the propagation is lower (less area burned). There is an interaction between the prob and the nint (see Figure/Comments below).

if ptype = "radius", then length of the radius in number of cells

if ptype = "all", then ppar is ignored.

if ptype = "hier", then it is a matrix with x and y levels and probabilities (in this order: first, second and third column) for hierarchically structured random landscapes.

if ptype = "matrix", then it is a matrix (lass object) or a vector with the names of the different matrices. Matrices are 0 for the non-perturbed and any other number for disturbed. If more than one matrix is included, but the number of disturbances is higher than the number of matrices, then the matrices are recycled, that is, it starts again for the first matrix as many times as necessary.

if ptype = "fuel", then ppar is the maxim proportion of landscape able to burn (by default = 100%). If 2 values are included, the first in the mean and the second de CV (default ppar = c(100, 0)). This can be used to limit the fire size.

pscope – Area in which disturbance may be propagated (by default all landscape). This is a matrix of 0 (no propagation, e.g. unburneable area) and 1 or non-zero (propagation is possible; any number different from 0 will behave as a 1). If not included, then disturbance can be propagated to all landscape, depending on the other propagation options.

Examples:

(see more examples in the Fateland model)

```
rfirehigh <- disturbance(0.05)
rfirelow <- disturbance(0.01)
# similar, but slightly different fire regimes are:
rfirehigh2 <- disturbance(tpar= c(20, 10), "syst")
rfirelow2 <- disturbance(tpar= c(100, 10), "syst")

xy <- matrix.col(x= c(1, 2, 3, 4), y= c(10, 20, 30, 40))
rfire2 <- disturbance(0.05,"rand", stype= "syst", loc= xy)

fsys20surface <- disturbance(20, "sys", ptype = "all", stratum= 1) # all
```

cells disturbed every 20 yrs, but affecting only the lower strata

```
# creating disturbance patterns
hmat <- matrix.col( xl= c(2,5,5), yl= c(2,5,5), p= c(1,0.5,0.9) )
distmat <- pattern(matrix(0, 50, 50), 1, "hier", hmat1)
mydist1 <- disturbance(0.05, "rand", ptype= "matrix", ppar= distmat)
mydist2 <- disturbance(0.05, "rand", ptype= "hier", ppar= hmat)
```

landscape

brolla – generates an object with the characteristics of the landscape

landscape(nr, nc, psize= 10.00, kluz= 2.50, wd= "n", wi= "none", topo= "", maxbiom= 40.00, r= 10.00, soildepth= 100.00)

nr – number of rows of the gridded landscape
nc – number of columns of the gridded landscape
psize – pixel size (real value); length of the side of the squared pixel (def = 10)
kluz – light extinction coefficient (default 2.50)
wd – direction of the dominant wind (def = NILL, no wind): "n", "ne", "e", "se", "s", "sw", "w", "nw"
wi – intensity of the wind ("none", "low", "med", "high").
topo – name of the terrain (topography) matrix objects (topo = "mytopo")
maxbiom – Maxium biomass in a cell. It can be one generic value for all cells, or a matrix of the same size to the landscape with a value for each cell. (default 40)
r – number of soil layers (default 10.00)
soildepth – Soil depth of the cell (in cm). It can be one generic value for all cells, or a matrix of the same size to the landscape with a value for each cell. (default 100 cm)

run

brolla – Executes the simulation for t years. `run` can be used with a graphic user interface (GUI) by setting the grid or plot options to TRUE (in this case, t is not needed). The run can be stopped/aborted using Ctl-Q (when running in non-graphical interface), etc....
run.load loads a previously saved binary file (see save2 option) as an initial condition for the run, and inserts the necessary species, landscape and disturbance lists.

run(t, land, sp, dist, tstep=1, plot=F, grid=F, save, save2, savet, savef=F)

run.load(t, file="", tstep=1, plot=F, grid=F, save, save2, savet, savef=F)

t – number of years
land – landscape objects generated by `landscape()`
sp – a vector with the name of the species object to be included. Species objects are generated by `species()`. Species list is already included in the landscape object (typically included with `distribution`), and so this option is solely needed when a species subset is desired in the run.
dist – the name of a disturbance object or a vector with the name of several disturbance objects. Disturbance objects are generated by `disturbance()`. If not included, no disturbance.
tstep - time step for displaying and saving the results (default, tstep=1, annual time step). If disturbance is applied, then the disturbance year is also displayed. That is, if only disturbance years are desired, then use `tstep` >= `t`.
grid, plot – FALSE (default)/TRUE for the use of a graphic interface. For using run in a source file, these options need to be FALSE.
save – a save option or a vector of save options. Save options in the assigned results object in run:
- "total": counts total, number of cells occupied each year (incl mat, imm, seeds, ..., disturbances), basal area (m^2/m^2), biomass (xx) and foliar area (xx) (Default)
- "spp": counts, basal area, biomass by species (as above for each species) for immature and mature plants
- "dist": count the number of cells for each disturbance type.

save2 – additional saved objects that can be generated during the run. It correspond to one of the options below, or to a vector with several options. The name of the resulting object will be the one given to assign the run results (maximum 8 characters), followed by "_" the abbreviated name of option (topv, rich, drec, ...) and the simulation year in which the save was performed (e.g. res_topv_50). In the case of the species matrices, the name of the species (in the field $name of the species object) is used instead of the option (res_Quercus_50). Options are:
- "sppba": matrix (map) for each spp with basal area values (xx?).
- "sppbio": matrix (map) for each spp with biomass values (xx?).
- "sppfa": matrix (map) for each spp with foliar areas values (xx?).
- "sppn": matrix (map) for each spp with the number of plants.
- "topview": a matrix indicating the dominant species in each cell (the species with highest basal area or biomass xxx?). If it is a fire year, then burned cells are indicated with a negative value.
- "richness": a matrix with the number of species in each cell.
- "distrec": disturbance recurrence. A matrix (map) with the number of disturbances in each cell.
- "dist": disturbance pattern (0/1) (it saves as many maps/matrices as disturbances). Matrices are stored with the name assigned to the run results followed by "_dist_X" where X is the simulation year (in this case, savet is not considered).
- "xyall": a matrix with the columns: row, column, species, number of individuals, basal area, seedpool???. Note that this can be a very large matrix.
- "binaryland": save the whole landscape in binary form to a file. It can be reloaded by run.load() . The name of the file will be the name assigned to

the run results followed by "_binaryland_X" where X is the simulation year. It can also be saved from the GUI (menus).

savet – (savetime) a value or vector indicating the simulation year in which save2 is performed. A -1 (default) means to save at the end of the simulation. Eg: `$savet <- seq(0, 100, by= 10)` would save data every 10 years. `$savet <- c(0, 50, -1)` would save data before starting the simulation, at year 50, and at the end. The vector should be sorted, except for –1 that should be the last value (otherwise, the save process will stop after the –1).

savef – (saveToFile) By default (savef=F) the save2 options are stored in the lass environment. By setting savef=T, the information on save2 is stored in ascii files (comma-separated) in the current working directory; these files can be read by many programs; lass can read them using `read("filename", sep=",")`. savef does not apply to "binaryland" which is always saved as a file in binary format.

file – Name (and path, if it is not in the working directory) of the binary file saved using save2= "binaryland" (or from GUI menus). It is used for running the model from the initial conditions of a previous run. This file contains all the information for running Brolla, including landscape, species and disturbance lists.

Example:

species, species.edit

brolla – set the species parameters for the brolla model by generating a list type object. `species()` generates a species variable (for each species). `species.edit()` is the same as `species()` but with a graphic interface; it also shows the shape of the dispersal function depending on the parameters chosen `species.edit()`.

species (name, maxdiam, maturediam, maxage, maxheight, seedlong= 1.00, shadetol= "med", droughttol= "med", resp= 0, lf= "pinus", firestim= 1, col= "", g= -1, sdisp= "no", mdisp= "no", ldisp= "no", kdisp= 2.0, limits= 100, fecund= 5, wind= 0, rgrowth= 0.0, fuel= 50).

species.edit( )
species.edit(sp)

name – species name (a string). Blank spaces and other reserved characters (-,+,/,=,*,:,$,<,>, ...) are ignored; underscore '_' is accepted.
maxdiam – maximum diameter (in cm)
maturediam – diamter at which plant reach maturity (in cm)
maxage – maximum age (in years)
maxheight – maximum height (in cm)
seedlong – seed longevity in years (defaul = 1)
shadetol – Shade tolerance, one of the following: "low", "med", "high"(defaul = "med")
droughttol – Drough tolerance, one of the following: "low", "med", "high"(defaul = "med")
resp – Resprouting capacity ... defaul = 0.00)
lf - Life form; one of the following: "pinus", "herb", "SprWoody", "NonSprWoody" (defaul = "pinus")

firestim – Multiplicative factor fot the post-fire germination, that is, 1 (no estimulation, i.e. germination as without fire), 2 (doble germination), etc... 0.5 (half germination), 0 (no post-fire germination), etc... (defaul = 1)

g – Growth rate; if –1 them it is computed following the Botkins et al. equiation (defaul = -1)

col – basic colour for the species in the grid expressed as RGB (vector of 3 values between 0 and 255). Final colours will change (a gradient of this basic colour) for immatures and for abundance (low, med, high) of matures. Currently it needs to be included as a column, e.g., t(c(255,0,0)), (i.e., red.)

sdisp – short-distance dispersal capacity: "no", "high", "med", "low" (def= med). Short-distance dispersal is a randomly uniform probability of dispersion from the cell to the first limit of the parameter *limits*. Values are 0.9, 0.6 and 0.3 (and currently they cannot be modified).

mdisp – medium-distance dispersal capacity: "no", "high", "med", "low" (def= med). These parameters (high, med, low) are related to the initial dispersal probability (`sdisp`) of the decay function, from the first limit to the second. The actual dispersal parameter depends also on the decay rate (`kdisp`). Prob. of dispersal = med * exp(-kdisp * (distance-limits[1]) / limits[2]) where distance is chosen randomly and med is 0, 0.3, 0.6, 0.9 for "no", "high", "med", "low" mdisp.

ldisp - long-distance dispersal capacity: "no", "high", "med", "low" (def= med). This is a uniform function (as in the short distance one), with a dispersal probability equal to the minimum of the decay function for medium-dispersal for the same class.

kdisp – decay rate for medium-distance dispersal. [def= 2]. Higher kdisp (lower -kdisp) implies lower probability of dispersal for any distance (see parameter med). If kdisp= 0, then the dispersal probability is the same at any distance and equal to the med value (0, 03, 0.6, 0.9 for no, low, med, high, respectively). It can be a vector with two elements, the mean and the CV.

limits – 2 or 3 values (in m) referring to the limit between short and medium dispersal, medium and long dispersal, and the maximum dispersal distance. Default values are 10, 100, and D (D= the diagonal of the landscape). Note that if limits[1]= 0 and limits[2]=limits[3] then only the decay dispersal of the medium-dispersal is applied. If uniform dispersal with distance is sought, then limits[2] and limits[3] should be the same. If limits[3] is not included, then D is assumed.

fecund – Number of distances randomly chosen each year; that is, the maximum viable "seeds" dispersed. It is a measure of the species fecundity (amount of seeds produced). fecund= 0 would produce no dispersal, independent of sdisp, mdisp and ldisp.

fecundfire – for serotiny species, the additional post-fire fecundity. One or two values (mean and coeficient of variation).

fecundmast – replace "fecund" with the given frequency. A vector of 5 values: fecundiy value; CV; interval; CV; initial. Intial is the year to start the masting (syncrhonic masting), and if it is negative, then masting starts in a different moment for each cell (non-synchronic).

wind – if wind= 0, dispersal is not affected by the wind (wind-independent e.g., animal-dispersed only), otherwise wind has effects; See `landscape()`.

fuel – (optional) combustibility index as %; default = 50%

Examples:

References

[Download Pausas publications from: http://www.ceam.es/lass/papers.htm]

Cliff, A.D.& Ord, J.K. 1981. Spatial processes: models and applications. Pion, London.

Cressman, G.P. 1959. An operational objective analysis system. Monthly Weather Review 87: 367-374.

Getis, A. & Ord, J.K. 1992. The analysis of spatial association by use of distance statistics. Geographical Analysis 24: 189-199.

McGarigal, K.& Marks, B.J. 1994. FRAGSTAT: Spatial pattern analysis program for quantifying landscape structure - Version 2.0. Forest Science Department, Oregon State University.

Moore, A.D. & Noble, I.R. 1990. An individualistic model of vegetation stand dynamics. J. Environ. Manage. 31: 61-81.

Ord, J.K. & Getis, A. 1995. Local spatial autocorrelation statistics: distributional issues and an application. Geographical Analysis 27: 286-296.

Pausas, J.G. 1999. The response of plant functional types to changes in the fire regime in Mediterranean ecosystems. A simulation approach. J. Veg. Sci. 10: 717-722. [pdf]

Pausas, J.G. 2003. The effect of landscape pattern on Mediterranean vegetation dynamics – A modelling approach using functional types. J. Veg. Sci. 14: 365-374. [pdf]

Pausas J.G. & Ramos J.I. 2005. Landsscape Analysis and Simulation Shell (LASS). Environmetal Modelling and Software, in press. [pdf]

Contacts, download and acknowledgements

- Currently LASS can be freely downloaded for research and educational purpose from: http://www.ceam.es/lass.
- The development of the LASS software has been financed by the projects SPREAD (EVG1-CT-2001-00043; European Union) and INVASORAS (REN2000-0361, Spanish Government).
- Contact: Juli G. Pausas (juli at ceam dot es).