Spatially explicit grazing raises risk of catastrophic shifts in drylands

by Florian Schneider and Sonia Kéfi (Institut des Sciences de l'Evolution de Montpellier (ISEM), CNRS, IRD, University of Montpellier 2)

Contents

This repository contains the original source code for a simulation study within the CASCADE project.

Project outline

Spatial models of vegetation cover so far have considered grazing mortality a rather constant pressure, affecting all plants equally, regardless of their position in space. In the known models it usually adds as a constant to the individual plant risk (Kéfi et al 2007 *Theoretical Population Biology*, 71:367–379). However, grazing has a strong spatial component: Many plants in rangelands invest in protective structures such as thorns or spines, or develop growth forms that reduce their vulnerability to grazing. Therefore, plants growing next to each other benefit from the protection of their neighbors.

Such **associational resistance** is widely acknowledged in vegetation ecology but hardly integrated in models as a cause for spatially heterogenous grazing pressure. It also

renders the plant mortality density dependent, which has important impacts on the bistability of the system.

We investigate how the assumption of spatially heterogeneous pressure alters the bistability properties and the response of spatial indicators of catastrophic shifts.

approach

Over a dual gradient of environmental and grazing pressure, we simulate the steady state of vegetation if starting from high vegetation cover. Complementary, we simulate how likely a degraded landscape is to restore if only few plants are left. The overlap of the vegetated state and the persistent desert is the domain of bistability.

Besides vegetation cover, we investigate which patterns of vegetation establish under the different types of pressure.

main findings

Code

simulation functions (`simfunctions.r`)

count()

usage:

```
count(x, neighbor)
```

parameters:

- `x` : the landscape object to be counted
- `neighbor`: the state of the cells to be counted in the neighboring cells

the function returns a vector with one integer value for each cell of the lattice. This value represents the number of neighbors in state `neighbor` for each single cell. Division by 4 gives the local density of cells in this state.

mapping()

usage:

```
mapping(width, height, boundary = "periodic",
     i_matrix = matrix(c(0,1,0,1,NA,1,0,1,0), ncol = 3, byrow = TRUE))
```

parameters:

- `width` & `height` : dimensions of the grid. This must match the width and height of the landscape objects that are later provided to the count function.
- `boundary`: default and only implemented option is "periodic", which means that a cell at the left border of the grid shares an edge with the cells on the right border of the grid, cells at the bottom share an edge with cells on top. This results in a borderless behaviour of the automaton.
- `i_matrix`: the interaction matrix to be assumed for the cellular automaton. In this matrix, the position with the value NA gives the position of the focal cell. The neighboring cells to be taken into account in the assessment (using the `count()` function) take value 1. Cells with value 0 are not taken into account. The default is the 4-cell neighborhood (von Neumann-neighborhood of range 1).

the function creates mapping vectors in the R global environment: `x_with_border` allows to translate the landscape object, which contains a row-wise vector of the cell states, into an extended vector that includes the neighboring cells at the border. `x_to_evaluate` is used to revert the transformation. Both maps are used in the count function to vectorize the calculation of local densities for reasons of calculation speed.

patches()

usage:

```
patches(x, state)
```

parameters:

- `x` : the landscape object to be evaluated
- `state` : the state of cells to be evaluated as patch, can contain a character vector with multiple states

The function uses an iterative process to identify all connected areas on the lattice that are of state `state` and are connected by at least one edge, *i.e.* a patch. The function returns a vector of individual patch sizes (number of cells).

fitPL()

usage:

```
fitPL(psd, p_spanning, n = NULL)
```

The function is quite specific and requires refinement to be re-used in other projects! It requires a valid object `psd` which is a `data.frame` containing cumulative patch-size distributions, i.e. a table with a column called `s` with the particular sizes occuring in the landscape, and a column called `p` with the probability of any patch being equal or larger than that size. The object `psd` can contain pooled data from multiple landscapes (combined into one data.frame using `rbind()`).
The function fits three alternative cumulative patch-size distribution functions, a limited power-law (up-bent), a straight power-law, and a truncated power-law (down-bent).
The returned object is a list with the entries TPLdown, PL, TPLup, containing the respective model outputs, as well as AIC, dAIC and best, which contains the AIC of the models, the delta AIC in respect to the lowest AIC value, and the ID number of the best model (2 = truncated power-law; 3 = straight power-law; 4 = limited power-law).

simulation code

template simulation code (`simulation.r`)

This code is the core implementation of a cellular automaton with 'local facilitation' and 'associational resistance'. It can be used to explore the parameter range manually.

The code contains a switch for associational resistance. If `parameters$assoc == FALSE` the grazing mortality still depends on the global vegetation cover, i.e. a mean field assumption on associational resistance.

simulation of the vegetated state (`sim_vegetated.r`)

This is the original simulation code used to produce the results of the study. It initialises a list of parameter combinations `iterations`, that iterates environmental quality, b (a sequence from 0 to 1 with a steplength 0.02) and grazing pressure (a sequence from 0 to 0.5 with a steplength of 0.01), which is used to invoke instances of the simulation code on a parallel cluster, using `foreach() %dopar%` of the foreach package (see below).
Each parameter combination is replicated 5–10 times on a landscape that is initialized with randomly distributed plants. The initial vegetation cover is drawn as a uniform random number within the range of 0.8 and 0.9. This simulation serves to evaluate the steady state vegetation cover and spatial pattern arising from each parameter combination.

The result summary that is returned in `result$out` contains mean values of these replicates. Also the cumulative patch size distributions calculated from the final landscapes of the replicates are pooled into an object `dd4` and fitted using the function `PLfit()` (see above).

The lines stored in `result$out` of all parameter sets are merged into one data.frame by the `foreach()` function and stored into a file `output.csv`.

sumulation of the recovery from low vegetation cover (`sim_desert.r`)

As above, but the simulation is replicated 100 times on a landscape with a vegetation cover of 0.001, *i.e.* 10 randomly distributed plants. The simulation runs only over max. 100 years (less if the landscape falls to a cover of 0), and no spatial structure is assessed. The code returns the probability for each parameter combination that the landscape recovers to at least a cover of 0.01 (100 plants) within 100 years.

simulation of the envelope of homogeneous grazing (`sim_bifurcation.r`)

This simulation code complements the simulation of associational resistance with the assumption of homogenous mortality on plants that are invulnerable to grazing, or on plants that are all equally vulnerable to grazing. It only runs over two sections along the gradient of grazing pressure ($g = 0.1$ and $g = 0.4$). See paper for details.

Parallel backend requirements

The function `foreach()` (of the R package foreach) that evokes the simulation for each parameter set makes use of a parallel backend, but falls back to sequential execution if none is provided. See the package documentation. For instance, the libraries doSNOW and snow can provide a parallel backend in R.

License