Simulations of livestock resilience model with caspr

Florian D. Schneider

October 1st, 2015

The livestock resilience model

The model was designed as a simplification of our previously developed grazing model. Besides reflecting the mechanisms of positive local feedbacks, local facilitation and associational resistance, it also implements negative local feedbacks, such as competition and attractant-decoy effects.

Caution: The model is still preliminary! Right now, it responds to landscape size, which should not be the case. I need to refine the parameters of the livestock functional response (a, h and q) to be independent of spacial extent of the lattice.

parameters

(More model details will follow)

The caspr package

Installation

Installation instructions and further information can be found on the package repository website on Github and the package vignette.

In short, you only need a running version of R installed on your computer. All package dependencies will be installed automatically. Launch R and run:

```
install.packages("devtools")
devtools::install_github("fdschneider/caspr")
```

Then, once installed, you will have to call the library each time you start a new R session

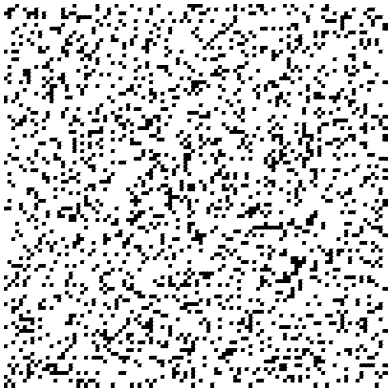
```
library(caspr)
```

run simulations

The package is capable of running a couple of different models. For our project, I added the livestock resilience model to the package. You can review it's parameters and specifications by typing `?livestock`.

To run a model simulation, you have to provide an initial landscape object, i.e. a grid of cells that matches the specifications of the model.

```
l <- init_landscape(states = c("1", "0"), cover = c(0.2,
    0.8), width = 100)
par(mar = c(0, 0, 0, 0))
plot(l)
```

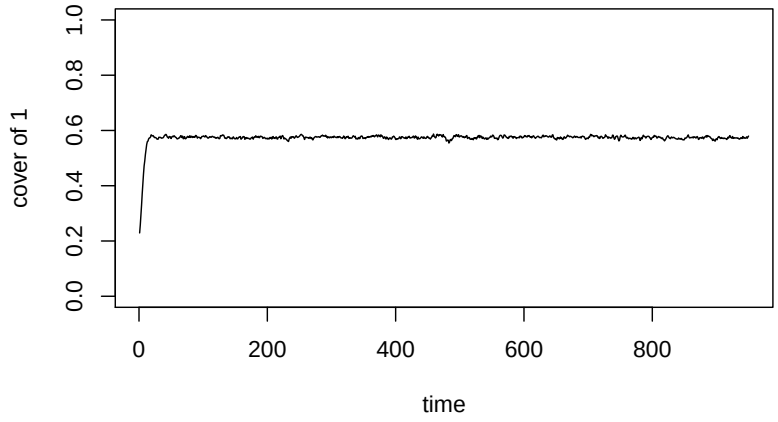


The option `width` allows to increase or decrease the landscape size (now $100x100$ cells). Then, you can start a simulation using the default parameters (returning a warning at the end of the simulation) with

```
run <- ca(l, model = livestock)
```

```
## Warning in ca(l, model = livestock): you did
## not specify 'parms'! using default parameters
## of model!
```

```
plot(run)
```




```
summary(run)
```

```
## $name
## [1] "Livestock resilience model"
##
## $time
## [1]   1 950
```

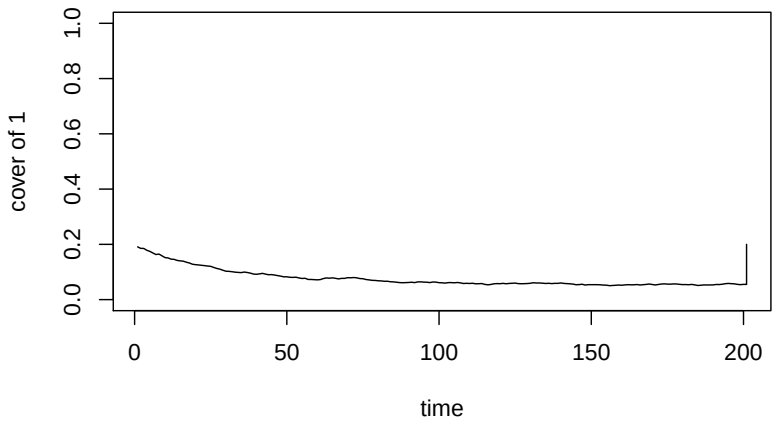
```
##
## $mean_cover
##         1         0
## 0.5734507 0.4265493
##
## $sd_cover
##           1           0
## 0.004206354 0.004206354
##
## attr(,"class")
## [1] "ca_summary"
```

The function `ca()` runs the simulation. This might take between 10 seconds and a couple of minutes, depending on your computer.

The `plot()` function plots the timeseries of the vegetation cover (cover of cell state "1") and the `summary()` function returns the mean cover of all potential cell states.

Of course, you will want to modify the parameters (remember that `?livestock` shows you the description of all parameters):

```
pp <- livestock$parms
pp$b <- 0.1
pp$p <- 0.2
pp$L <- 12
run <- ca(l, model = livestock, parms = pp, t_max = 200)
plot(run)
```



`t_max` sets the maximal number of timesteps. More options for the function `ca()` are available by calling `?ca`.

access the simulation data

If you want to analyse the timeseries of vegetation cover you can extract it from the output object of the simulation, which is now stored in the R variable `run`.

For example you can access the parameters that were used to run the model by calling

```
as.data.frame(run$model$parms)
```

```
##   r   b   f alpha   K   c    m v   p  L q  h
## 1 1 0.1 0.9     0 0.8 0.1 0.05 0 0.2 12 0 50
##     a
## 1 0.2
```

You also can extract the vegetation cover for each single timestep and save it into an R object. To analyse variation over time, we usually would only look at a period before the end of the simulation.

```
t <- tail(run$time, 100)
cover <- tail(run$cover[["1"]], 100)
```

```
summary(cover)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
## 0.05050 0.05405 0.05600 0.05755 0.05832
##    Max.
## 0.20000
```

There are further functions currently under development that allow to run a whole array of parameters. I will keep you updated about this.

contribute

You always can check the current state of the development for the caspr package and report issues or bugs on the GitHub issue tracker.