demoniche

Hedvig Nenzén

April 10, 2012

Contents

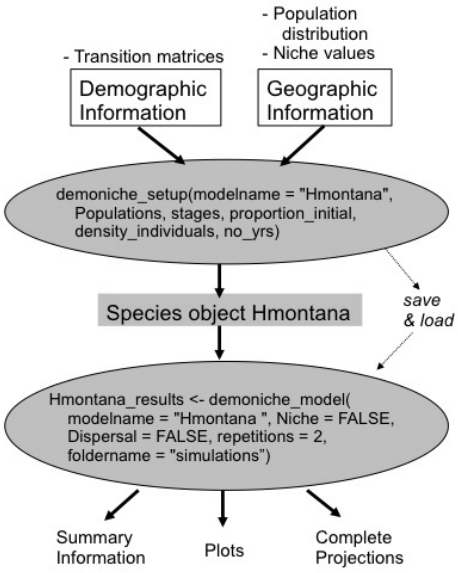

Figure 1: Simplified schema of the demoniche framework. First the user enters demographic and geographic information into the setup function. The model function runs the demographic modelling on the species object.

1 Introduction

`demoniche` is a freely available R-package to simulate stochastic population growth for various subpopulations of a species. Demographic models projects population sizes with various transition matrices that represent demographic impacts on species growth. The Demographic modelling is linked to a time series of geographically distributed 'Niche values' that also affect species growth. The demoniche model offers flexible options for stochasticity, density dependence and dispersal. With the demoniche package it is possible to investigate population sizes, extinction probabilities and range shift of a species influenced by scenarios of environmental and human impacts.

The `demoniche` package is offered without any guarantees, and we encourage users to examine the code to learn what is modelled, and to adapt it to their own needs. The annotated code can be found when browsing the subversion repository on R-forge (under the SCM tab).

The main steps to running a model are as follows:

- Load or write the information (demographical, geographical) of the species that is being modelled
- Create a species object with `demoniche_setup` function, which contains all the information about the species
- Run the `demoniche_model` function on the species object
- Analyse the results

2 How to use demoniche

First install R. We recommend that users be familiar with the R environment. Help can be gained from, for example www.cran.r-project.org.

2.1 Install the package

To install the demoniche package please type install.packages("demoniche", repos="http://R-Forge.R-project.org") in R. Or go to the webpage http://demoniche.r-forge.r-project.org/ and follow the directions. The package and manual are frequently updated, so please download an updated package often. This manual is available at http://demoniche.r-forge.r-project.org/pics/demoniche_manual.pdf.

The demoniche package depends on functions and data from three other packages; popbio, lattice and sp that are available from CRAN. They should install automatically, please do it manually if not.

Set the working directory, and then load the package. This makes the two main functions demoniche_model and demoniche_setup available (Figure 1). The demoniche_model function runs two internal functions, demoniche_population that carries out demographic modelling in each population and year, and demoniche_dispersal that calculates the dispersal if selected.

```
> library(demoniche)
```

The code used in this manual is also provided in the demoniche_manual.R script in the /doc folder where the package is saved, normally this is at C:\Program Files\R\R-2.13.0\library in Windows, or a similar location.

2.2 Loading data supplied with package

We load the example data file supplied in the package. The object is called Hmontanta and contains demographic and geographic data about Mountain Goldenheater *Hudsonia montana* (Gross et al. 1998). We can inspect the object with str(). We find that Hmontana is a list with 26 items. We can examine separate items of the list using $. This object contains all the information needed about the species to carry out modelling.

```
> data(Hmontana)

> str(Hmontana)

List of 27
 $ Orig_Populations          :'data.frame':        78 obs. of  4 variables:
  ..$ PatchID        : int [1:78] 8000 8001 8002 8003 8004 8005 8006 8007 8008 8009 ...
  ..$ XCOORD         : int [1:78] 2 3 4 5 6 7 8 10 11 12 ...
  ..$ YCOORD         : int [1:78] 24 24 24 24 24 24 24 24 24 24 ...
  ..$ area_population: int [1:78] 2 2 2 2 2 2 2 2 2 2 ...
 $ Niche_ID                  :'data.frame':        480 obs. of  4 variables:
  ..$ Niche_ID    : int [1:480] 5420 5421 5422 5423 5424 5425 5426 5427 5428 5429 ...
  ..$ X           : int [1:480] 1 2 3 4 5 6 7 8 9 10 ...
  ..$ Y           : int [1:480] 15 15 15 15 15 15 15 15 15 15 ...
  ..$ PopulationID: num [1:480] 0 0 0 0 0 0 0 0 0 0 ...
 $ Niche_values              : num [1:480, 1:4] 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:480] "1" "2" "3" "4" ...
  .. ..$ : chr [1:4] "period_2000" "period_2010" "period_2020" "period_2030"
 $ years_projections         : chr [1:4] "period_2000" "period_2010" "period_2020" "period_2030"
 $ matrices                  : num [1:36, 1:5] 0.4995 0.0004 0 0 0 ...
  ..- attr(*, "dimnames")=List of 2
```

```
  .. ..$ : NULL
  .. ..$ : chr [1:5] "Reference_matrix" "Matrix_1" "Matrix_2" "Matrix_3" ...
 $ matrices_var              : num [1:36, 1] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 ..
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "sd"
 $ prob_scenario             : num [1:2] 0.5 0.5
 $ noise                     : num 0.95
 $ stages                    : chr [1:6] "seed" "seedlings" "tiny" "small" ...
 $ proportion_initial        : num [1:6] 0.98181 0.000691 0.006908 0.003684 0.005756 ...
 $ density_individuals       : num [1:78] 20000 20000 20000 20000 20000 20000 20000 20000 20000 2
 $ fraction_SDD              : num 0.05
 $ dispersal_probabilities   : num [1:480, 1:480] 0 0.258 0.232 0.217 0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:480] "5420" "5421" "5422" "5423" ...
  .. ..$ : chr [1:480] "5420" "5421" "5422" "5423" ...
 $ dist_latlong              : num [1:480, 1:480] 0 1 2 3 4 5 6 7 8 9 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:480] "5420" "5421" "5422" "5423" ...
  .. ..$ : chr [1:480] "5420" "5421" "5422" "5423" ...
 $ neigh_index               : num [1:2] 1 1.4
 $ fraction_LDD              : num 0.05
 $ no_yrs                    : num 10
 $ K                         : num 10000
 $ Kweight                   : num [1:6] 0 1.5 1 1 1 1
 $ populationmax_all         : num [1:480, 1:4] 10000 10000 10000 10000 10000 10000 10000 10000 1
 $ n0_all                    : num [1:480, 1:6] 0 0 0 0 0 0 0 0 0 0 ...
 $ list_names_matrices       :List of 5
  ..$ : chr "Reference_matrix"
  ..$ : chr "Matrix_1"
  ..$ : chr "Matrix_2"
  ..$ : chr "Matrix_3"
  ..$ : chr "Matrix_4"
 $ sumweight                 : num [1:6] 0 1 1 1 1 1
 $ transition_affected_env   : int [1:24] 1 2 9 13 14 15 16 19 20 21 ...
 $ transition_affected_niche : num [1:2] 1 3
 $ transition_affected_demogr: int [1:24] 1 2 9 13 14 15 16 19 20 21 ...
 $ env_stochas_type          : chr "normal"
NULL
```

```
> Hmontana$env_stochas_type
```

```
[1] "normal"
```

2.3 Running simulations

We use the modelling function `demoniche_model(modelname, Niche, Dispersal, repetitions, foldername)` to carry out the demographic modelling, and specifying the `Hmontana` list of species information as the species object we want to use. As arguments to the function the user also needs to specify if you want to run simulations with the effects of the Niche values (TRUE or FALSE) and if you want to allow long-distance dispersal (TRUE or FALSE). You also need to specify how many repetitions you want to carry out (for stochastic simulations the number should be over 1000), and a name for the folder where the simulations will be stored. But all these simulations are carried out with the same species information.

The `demoniche_model` function runs two internal functions, `demoniche_population` that carries out demographic modelling, and `demoniche_dispersal` which calculates the dispersal if selected.

When we run the `demoniche_model` function messages are printed on the screen, to let us know how the simulations are going.

```
> noCC_nodispersal <- demoniche_model(modelname = "Hmontana", Niche = FALSE,
+                          Dispersal = FALSE, repetitions = 2,
+                          foldername = "noCC_nodispersal")

[1] Starting projections for repetition: 1
[1] Projecting for scenario/matrix: Reference_matrix
[1] Projecting for scenario/matrix: Matrix_1
[1] Projecting for scenario/matrix: Matrix_2
[1] Projecting for scenario/matrix: Matrix_3
[1] Projecting for scenario/matrix: Matrix_4
[1] Starting projections for repetition: 2
[1] Projecting for scenario/matrix: Reference_matrix
[1] Projecting for scenario/matrix: Matrix_1
[1] Projecting for scenario/matrix: Matrix_2
[1] Projecting for scenario/matrix: Matrix_3
[1] Projecting for scenario/matrix: Matrix_4
[1] Calculating summary values
[1]     All repetitions completed!
```

To change any of the parameters, we simply change the arguments in the function, and the `foldername`, to change where the objects are saved. Here we have chosen to include effects of Niche values but no dispersal.

```
> CC_nodispersal <- demoniche_model(modelname = "Hmontana", Niche = TRUE, Dispersal = FALSE, repe

[1] Starting projections for repetition: 1
[1] Projecting for scenario/matrix: Reference_matrix
[1] Projecting for scenario/matrix: Matrix_1
[1] Projecting for scenario/matrix: Matrix_2
[1] Projecting for scenario/matrix: Matrix_3
[1] Projecting for scenario/matrix: Matrix_4
[1] Starting projections for repetition: 2
[1] Projecting for scenario/matrix: Reference_matrix
[1] Projecting for scenario/matrix: Matrix_1
[1] Projecting for scenario/matrix: Matrix_2
[1] Projecting for scenario/matrix: Matrix_3
[1] Projecting for scenario/matrix: Matrix_4
[1] Calculating summary values
[1]     All repetitions completed!
```

2.4 Analyse results

From running the `demoniche_model` function we get these outputs:

- Summary statistics from all the repetitions in the workspace (as value of function)
- Plots of EMA (Expected Minimum Abundance, McCarthy and Thompson 2001), population sizes and patch occupancy
- Population simulation results and EMAs saved in a in csv file and as R objects (.rda)

- Population data, Eigen analysis, EMAs, saved in folder as R objects

First, the output from the `demoniche_model` function itself is an array containing the population sizes at each time step (mean, standard deviation of mean population sizes, minimum and maximum), calculated from all repetitions. If you only have one transition matrix, and run models without stochasticity, the mean the standard deviation will be zero, regardless of numbers of repetitions. The third dimension of the array are the results for the different scenario matrices.

Complete yearly population stage distributions from each repetition and each population are saved in the folder with the specified name. These data can be visually or statistically analysed, or exported to other formats for further analysis of transient dynamics. The function also creates line graphs of the mean population results for each year, EMA of each transition matrix scenario scenario for each time period, and maps of the occupancy throughout the modelled area, at each time step (Fig. 2)

```
> dim(noCC_nodispersal)
```

```
[1] 40  4  5
```

```
> dimnames(noCC_nodispersal)
```

```
[[1]]
 [1] "year1"  "year2"  "year3"  "year4"  "year5"  "year6"  "year7"
 [8] "year8"  "year9"  "year10" "year11" "year12" "year13" "year14"
[15] "year15" "year16" "year17" "year18" "year19" "year20" "year21"
[22] "year22" "year23" "year24" "year25" "year26" "year27" "year28"
[29] "year29" "year30" "year31" "year32" "year33" "year34" "year35"
[36] "year36" "year37" "year38" "year39" "year40"

[[2]]
[1] "Meanpop" "SD"      "Max"     "Min"

[[3]]
[1] "Reference_matrix" "Matrix_1"         "Matrix_2"
[4] "Matrix_3"         "Matrix_4"
```

We can also look at parts of the object itself, here the mean populations from simulations with the scenario matrix 1:

```
> noCC_nodispersal[,"Meanpop","Matrix_1"]
```

```
  year1   year2   year3   year4   year5   year6   year7   year8   year9
52845.5 49425.0 46441.0 43814.5 41398.5 39237.5 37266.0 35398.5 33653.0
 year10  year11  year12  year13  year14  year15  year16  year17  year18
32029.0 30449.0 28910.5 27495.5 26104.0 24844.0 23618.0 22419.0 21300.0
 year19  year20  year21  year22  year23  year24  year25  year26  year27
20232.0 19222.5 18266.0 17319.0 16409.0 15573.0 14757.0 13954.5 13182.5
 year28  year29  year30  year31  year32  year33  year34  year35  year36
12449.5 11744.5 11089.5 10429.5  9823.5  9243.5  8671.5  8127.0  7611.0
 year37  year38  year39  year40
 7123.5  6634.0  6180.5  5742.5
```

The values obtained in other runs will be different as their population growth is stochastic.

We can plot the population sizes at year number 40. Figure 2 is produced by the following code:

```
> barplot(cbind(noCC_nodispersal[40,2,], CC_nodispersal[40,2,]), beside = TRUE,
+       legend.text = Hmontana$list_names_matrices, names.arg =
+       c("no Niche values","with Niche values"))
```

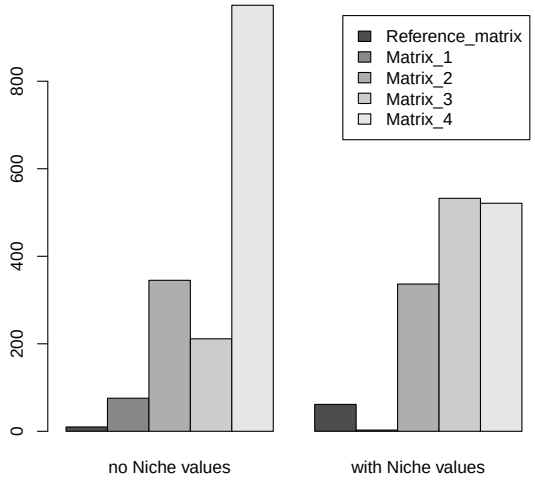

Figure 2: Population size (number of individuals) for simulations under each scenario matrix scenario of human land use, in the last year of simulation (year 90). With climate change there was a lower predicted population size.

Apart from the summary object, the results from each simulation are saved in the folder with the specified name, in the working directory. If any of the files are open while we run new simulations, they will not be updated (i.e. automatically overwrites files, but cannot do it if files are open). We can look at what results are called:

```
> list.files(path = "noCC_nodispersal")
```

```
 [1] "EMA.rda"
 [2] "EMAs.jpeg"
 [3] "Projection_rep1_Matrix_1.rda"
 [4] "Projection_rep1_Matrix_2.rda"
 [5] "Projection_rep1_Matrix_3.rda"
 [6] "Projection_rep1_Matrix_4.rda"
 [7] "Projection_rep1_Reference_matrix.rda"
 [8] "Projection_rep2_Matrix_1.rda"
 [9] "Projection_rep2_Matrix_2.rda"
[10] "Projection_rep2_Matrix_3.rda"
[11] "Projection_rep2_Matrix_4.rda"
[12] "Projection_rep2_Reference_matrix.rda"
[13] "eigen_results.rda"
[14] "map_Matrix_1.jpeg"
[15] "map_Matrix_2.jpeg"
[16] "map_Matrix_3.jpeg"
[17] "map_Matrix_4.jpeg"
[18] "map_Reference_matrix.jpeg"
[19] "metapop_results.rda"
[20] "population_results.csv"
[21] "population_results.jpeg"
```

```
[22] "population_results.rda"
[23] "population_sizes.rda"
```

In the folder there are various .jpgs that show the Mean population sizes (with +-1 standard deviation on each side in the same color if there are more than one repetition), EMA through time for all transition matrices, and the patch occupancy at each time period, for the last repetition. These are intended for a quick visualization of results. The original data can be accessed and plotted as the user wants.

Other results are saved in `.csv` which can be opened in for example excel, such as the `population_results.csv`. The rows are the information for each matrix. The columns contain information about the projects: initial total population area, initial population, percentage of populations that went extinct during the simulations, the growth rate (lambda, λ, the first eigenvalue) of the scenario matrix (not the reference matrix, function from `popbio` package), the stochastic lambda from the two matrices (50.000 time intervals, function from `popbio` package).

Other data, such as the complete occupancy for each population and each repetition, for all stages, are saved in other formats. The `.rda` format can be loaded with the command `load()` into R. This data can then be used for plotting, further analyzed, etc.

2.5 Sensitivity analysis

The user can also load the object `eigen_results_yourmatrixname.rda` for each transition matrix. It is a list with an entry for each matrix, which is made up of the results from the function `eigen.analysis` from the `popbio` package. The list contains the eigenvalue of the matrix (λ), the stable stage distribution, the sensitivities and elasticities, the reproductive value of each life stage, and the damping ratio. Other functions in the `popbio` package, such as `pop.projection()` can be used to determine if our transition matrix and data have initial transient periods, that we might have to deal with. For further explanation of the eigen analysis please see `popbio` package helpfiles, Morris and Doak, 2002, and other publications listed below.

```
> load('noCC_nodispersal/eigen_results.rda')

[1] "eigen_results"

> str(eigen_results)

List of 5
 $ Reference_matrix:List of 7
  ..$ lambda1      : num 0.959
  ..$ stable.stage : num [1:6] 0.98142 0.00075 0.00398 0.00342 0.00476 ...
  ..$ sensitivities: num [1:6, 1:6] 0.0113 12.9282 0 0 0 ...
  ..$ elasticities : num [1:6, 1:6] 0.00587 0.00539 0 0 0 ...
  ..$ repro.value  : num [1:6] 1 1148 2305 4340 5552 ...
  ..$ damping.ratio: num 1.4
  ..$ LTRE         : num [1:6, 1:6] 0 0 0 0 0 0 0 0 0 0 ...
 $ Matrix_1        :List of 7
  ..$ lambda1      : num 0.959
  ..$ stable.stage : num [1:6] 0.974 0.0008 0.0075 0.00653 0.00811 ...
  ..$ sensitivities: num [1:6, 1:6] 0.00964 11.07891 0 0 0 ...
  ..$ elasticities : num [1:6, 1:6] 0.00502 0.00462 0 0 0 ...
  ..$ repro.value  : num [1:6] 1 1150 2311 4287 4661 ...
  ..$ damping.ratio: num 1.44
  ..$ LTRE         : num [1:6, 1:6] 0 0 0 0 0 0 0 0 0 0 ...
 $ Matrix_2        :List of 7
  ..$ lambda1      : num 1.01
  ..$ stable.stage : num [1:6] 0.987575 0.000812 0.001046 0.000606 0.000291 ...
```

```
  ..$ sensitivities: num [1:6, 1:6] 0.00926 11.8108 0 0 0 ...
  ..$ elasticities : num [1:6, 1:6] 0.00458 0.00468 0 0 0 ...
  ..$ repro.value  : num [1:6] 1 1276 2699 4696 7653 ...
  ..$ damping.ratio: num 1.29
  ..$ LTRE         : num [1:6, 1:6] 0 0 0 0 0 0 0 0 0 0 ...
 $ Matrix_3        :List of 7
  ..$ lambda1      : num 0.845
  ..$ stable.stage : num [1:6] 0.979559 0.000598 0.006777 0.005622 0.004388 ...
  ..$ sensitivities: num [1:6, 1:6] 0.00389 4.48501 0 0 0 ...
  ..$ elasticities : num [1:6, 1:6] 0.0023 0.00159 0 0 0 ...
  ..$ repro.value  : num [1:6] 1 1153 2144 7201 22964 ...
  ..$ damping.ratio: num 1.26
  ..$ LTRE         : num [1:6, 1:6] 0 -0.000826 0 0 0 ...
 $ Matrix_4        :List of 7
  ..$ lambda1      : num 1.02
  ..$ stable.stage : num [1:6] 0.986585 0.000589 0.00127 0.000474 0.002607 ...
  ..$ sensitivities: num [1:6, 1:6] 0.0152 26.2875 0 0 0 ...
  ..$ elasticities : num [1:6, 1:6] 0.00746 0.00774 0 0 0 ...
  ..$ repro.value  : num [1:6] 1 1729 3522 4523 4990 ...
  ..$ damping.ratio: num 1.36
  ..$ LTRE         : num [1:6, 1:6] 0 -0.00176 0 0 0 ...
NULL
```

We can use the `popbio` package function `image2` to plot the Sensitivity matrix (Figure 3). We can see that the stage with the highest sensitivity is the stage from seed to seedling (12.928).

```
$mar
[1] 1 3 3 1

$xpd
[1] TRUE

NULL
```

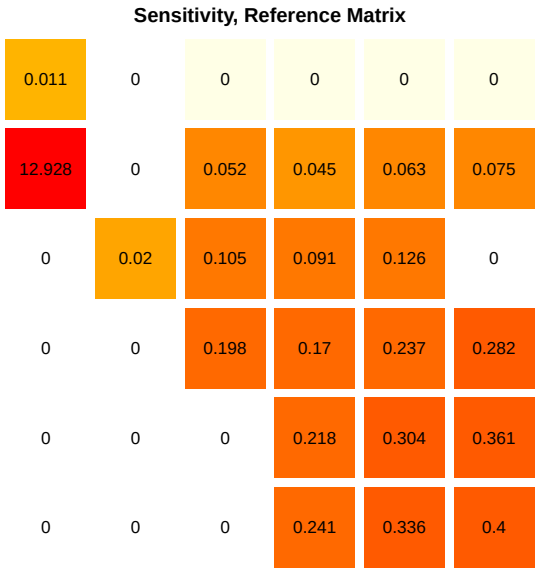

Figure 3: Sensitivities of the Reference Matrix for Mountain Goldenheather *Hudsonia montana*.

3 Parametrizing a model with our own data

To run simulations with data for our own species, we have to run the `demoniche_setup` function. We can examine the arguments that are required for the setup function and their defaults.

```
> args(demoniche_setup)
```

```
function (modelname, Populations, stages, Nichemap = "oneperiod",
    matrices, matrices_var = FALSE, prob_scenario = c(0.5, 0.5),
    proportion_initial, density_individuals, transition_affected_niche = FALSE,
    transition_affected_env = FALSE, transition_affected_demogr = FALSE,
    env_stochas_type = "normal", noise = 1, fraction_SDD = FALSE,
    fraction_LDD = FALSE, dispersal_constants = FALSE, no_yrs,
    Ktype = "ceiling", K = NULL, Kweight = FALSE, sumweight = FALSE)
NULL
```

The arguments that have no defaults (without '=') are `modelnames, Populations, stages, matrices, proportion_initial, density_individuals, no_yrs`. These arguments always require values that are defined by the user. The rest of the arguments have defaults (as shown) and do not need values. The defaults can be changed by the user, and below we describe the values that each argument can have (see section 3.4 for a minimal example). The `demoniche` model, and demographic modelling in general is 'data-hungry' and requires both detailed geographic and demographic information for the modelled species. For further discussion about how to obtain data, please see Morris and Doak (2002).

Overview of arguments to `demoniche_setup`:

Geographic information, 3.1

- 3.1.1 Information about occupied populations, with at least two sub-populations. Each patch must have coordinates, a (numeric) identifier, and a size (size can be the same for all patches if the model is grid-based).
- 3.1.2 A grid of present and future niche suitability values (Niche values), from 0-1 or 0-1000. They can be outputs from niche modelling (Species Distribution Modelling), scenarios derived from General Circulation Models (GCM), predictions of land use-changes, habitat cover types, topology, etc. They can also be any other values thought to influence the species growth (they can also be zeros if the surrounding environment is completely unsuitable for the species). If no Nichevalues are supplied, a background grid in which dispersal can occur is automatically created.

Demographic information, 3.2

- 3.2.1 Age(Leslie)- or stage(Lefkovitch)-based transition matrix. Various scenario or catastrophe matrices that represent a certain environmental effect (or threat). At each year in the simulation, either the mean matrix will be chosen, or the scenario matrix, with a user-defined probability, 3.2.12.
- 3.2.2 Name of each life stage
- 3.2.3 Weight of stages
- 3.2.4, 3.2.5 and 3.2.6 Which stages in the transition matrix which are affected by Niche values, and environmental and demographic stochasticity.
- 3.2.7 Type of distribution of the environmental stochasticity
- 3.2.8 Coefficients of variation in vital rates. Optional coefficient of variation matrix for each value in the transition matrix.

- 3.2.9 The initial proportion of individuals in each stage.
- 3.2.10 Initial population density.
- 3.2.11 Type of carrying capacity (K) to implement density dependent growth. The maximum population size for each patch, or a multiple above which the population cannot grow. The K can be set to vary both spatially and temporally.
- **??** Autocorrelation in matrix selection.
- 3.2.14 and 3.2.15 The fraction that seeds that undergo short and long distance dispersal.
- 3.2.16 Dispersal kernel

As an example of how we would load and model with our own data, we here re-create our example file `Hmontana` (Gross et al. 1998). We show how we could load our own information about a species, step by step. When we run `demoniche_setup` function on this information, we create the object `Hmontana`. This example can be run directly from the R-script `demoniche_manual.R` which is located in the `/doc` folder of the package folder. The package is usually saved in the R library folder, normally this is at `C:\Program Files\R\R-2.13.0\library` in Windows, or a similar location.

The information that we need to enter into the model can either be written into the workspace, or read from for example `.csv` files. They can either be defined as objects, or defined directly in the `demoniche_setup(...)` function. These objects can have any name, and in this example we have named them with the suffix `_mine` for clarity. Then, when running the setup function we specify which object corresponds to which argument in the function. For example, instead of `no_yrs_mine <- 10` and then defining the object in the setup function: `demoniche_setup(... no_years = no_yea` we can simply write the value `demoniche_setup(... no_years = 10)`. This is easier when values are short.

Each argument needs either an already defined object or a value, except those that already have a default.

3.1 Geographic information

3.1.1 Original population distribution

The `Populations` opbject should be one dataframe with ID of patches, coordinates of original locations, area of the population (in same unit as density data). If all populations have the same size (grid-based) the area will be the same for all populations. The data should be in lat/long format. The file can be downloaded from http://demoniche.r-forge.r-project.org/pics/Hudsonia_Populations_grids.csv

```
> Populations_mine      <-
+     read.table(file = "Hudsonia_Populations_grids.csv", sep = ";", header = TRUE)
> head(Populations_mine)

  patchID X  Y area
1    8000 2 24    2
2    8001 3 24    2
3    8002 4 24    2
4    8003 5 24    2
5    8004 6 24    2
6    8005 7 24    2
```

3.1.2 Niche data

The `Nichemap` object is the background map. It should be be a matrix with the niche values (some kind of habitat suitability values, islands, national parks, etc.), within the same geographical areas as the locations of the populations. The first column should be IDs, the second and third should be grid coordinates. The rest of the columns should be the different time periods that we wish to model.

We can also only demographic growth, without modelling a relationship between the habitat suitability and the population growth. In this case, `Nichemap` object should be a character vector of what our time periods should be called (i.e. `Nichemap <- c("2050", "2090")` if we want two time periods). The length of the time periods is set by the `no_yrs` argument (see below). If nothing is supplied, the default is one time period called "oneperiod". Within the setup function a background map is then created.

It is the niche data map resolution that determines the scale of the population sizes. Each population that falls within a grid of the niche data is spatially joined together to make one population (and their carrying capacities are summed). So using a niche map with a high resolution ensures that populations are kept separately, that each original population is maintaned within its own grid cell, and also that the maps are larger and use more memory. For memory maximization, all plots which will have a Niche value of zero at all time periods will be removed from the dataset that is modelled with, as they are unsuitable for the species anyway. The initial distribution of patches that is used for modelling can be inspected in `Hmontana$n0_all`.

In this example we can imagine that we are using predicted probabilities of presence under climate change (from a GLM). The file can be downloaded from http://demoniche.r-forge.r-project.org/pics/Hudsonia_SDMmodelling.csv.

```
> Nichemap_mine        <-
+         read.table(file = "Hudsonia_SDMmodelling.csv", sep = ";", header = TRUE)
> tail(Nichemap_mine)

    gridID  X  Y period_2000 period_2010 period_2020 period_2030
475   5894 25 30           1        0.95         0.9        0.85
476   5895 26 30           1        0.95         0.9        0.85
477   5896 27 30           1        0.95         0.9        0.85
478   5897 28 30           1        0.95         0.9        0.85
479   5898 29 30           1        0.95         0.9        0.85
480   5899 30 30           1        0.95         0.9        0.85
```

With the `lattice` package we can plot the Niche values and see how the most suitable areas move towards the south, Figure 3.1.2.

```
> niche_formulas <- as.formula(paste(paste(colnames(Nichemap_mine)[-c(1:3)],
+                       collapse="+"),"X+Y",sep="~"))

period_2000 + period_2010 + period_2020 + period_2030 ~ X + Y

> print(levelplot(niche_formulas, Nichemap_mine, col.regions=rev(heat.colors(100)),
+   main = "Niche Values"))
```

Length in years of each time period (each of the columns with niche values in the Nichemap is a time period).

```
> no_yrs_mine         <- 10

[1] 10
```

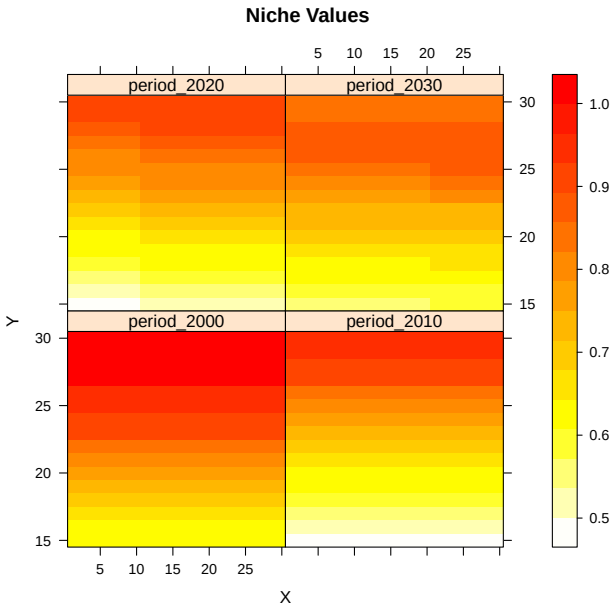

Figure 4: Niche values at each modelled time period.

3.2 Demographic information

3.2.1 Transition matrices

A matrix/data frame of transtion matrices for different scenarios, one column per matrix. The dimension should be a multiple of the number of stages. The first matrix should be the 'reference' matrix. The model can also run with a single matrix, in which case the mean population will be the same as the minimum and maximum population numbers, and the standard deviation NA (deterministic modelling).

Here we use data from the `popbio` package. The transition probabilities (matrix elements, a_{ij}) were originally in a square matrix format, so we unlist each matrix so that each matrix is now one column in the matrix object.

```
> library(popbio)
> data(hudvrs)
> data(hudsonia)
> matrices_mine <- cbind(meanmatrix = as.vector(hudmxdef(hudvrs$mean)),
+                                sapply(hudsonia, unlist))
> head(matrices_mine)
```

```
     meanmatrix    A85    A86    A87    A88
[1,]     0.4995 0.4995 0.4995 0.4995 0.4995
[2,]     0.0004 0.0004 0.0004 0.0003 0.0003
[3,]     0.0000 0.0000 0.0000 0.0000 0.0000
[4,]     0.0000 0.0000 0.0000 0.0000 0.0000
[5,]     0.0000 0.0000 0.0000 0.0000 0.0000
[6,]     0.0000 0.0000 0.0000 0.0000 0.0000
```

```
> colnames(matrices_mine) <- c("Reference_matrix", "Matrix_1", "Matrix_2", "Matrix_3", "Matrix_4"
```

We see that the `matrices` object is a matrix with each original transition matrix as a column and the names of the matrices as the column names (that do not agree with the original matrix names in this case).

The order of the transition probabilities is important. Make sure that running the function `matrix()`function with the option `byrow = FALSE` on one of the matrix columns reproduces the correct matrix. This depends on the original order of the matrix elements, if the matrix was constructed by row or by column. Below we see that the seed (fertility) stage is on the top row, which is correct. When later we select which transition probabilities stochasticity should effect, it is the row numbers in the matrix object that we use as a reference. For example, if we set effects on matrix entry 1, is refers to top left matrix element ([1,1]) of the matrix.

```
> matrix(matrices_mine[,"Reference_matrix"], ncol = 6, byrow = FALSE)
```

```
       [,1]   [,2]      [,3]       [,4]        [,5]        [,6]
[1,] 0.4995 0.0000 4.5782000 12.1425000 22.31670000 50.18950000
[2,] 0.0004 0.0000 0.0033000  0.0088000  0.01620000  0.03640000
[3,] 0.0000 0.4773 0.5988709  0.2025511  0.07961523  0.00000000
[4,] 0.0000 0.0000 0.1891171  0.4726192  0.10947094  0.06865331
[5,] 0.0000 0.0000 0.0000000  0.2662100  0.55730661  0.17653707
[6,] 0.0000 0.0000 0.0000000  0.0231487  0.24879760  0.73557114
```

3.2.2 Stages

In this case, the column names of the original matrix are used to make the `stages` object which has the names of each life stage.

```
> stages_mine <- colnames(hudsonia$A85)
```

```
[1] "seed"      "seedlings" "tiny"      "small"     "medium"
[6] "large"
```

3.2.3 Weight of stages

We can specify which life stages should be counted when calculating population sizes, and to which stages that the density dependent growth should be applied to. In this example, we do not count the seed stage when calculating population sizes, and the seeds are not subject to density dependent growth. It can also be "all_stages" if all stages should be included.

```
> sumweight_mine      <- c(0,1,1,1,1,1)
```

```
[1] 0 1 1 1 1 1
```

3.2.4 Stages affected by Niche values

Here we specify which transition probabilities (matrix elements, a_{ij})) should be affected by the Niche values. We can specify the character string "all" which means that all non-zero stages are affected. Alternatively, we enter the matrix stages as a vector of numbers which correponds to the correct matrix element. In this case, we set Niche values to affect rows 1 and 3 of the `matrices` object. We know that the first and third row in our `matrices` object refer to the probability of a seed surviving in the seed-bank and the probability of a seed becoming a seedling. Below we set which probabilities should be affected, and see what probabilities in the reference matrix they refer too.

```
> transition_affected_niche_mine <- c(1,3)
```

```
[1] 1 3
```

```
> matrices_mine[transition_affected_niche_mine,1]
```

```
[1] 0.4995 0.0000
```

3.2.5 Stages affected by environmental stochasticity

Here we specify which transition probabilities (matrix elements, a_{ij}) should be affected by and environmental and demographic stochasticity. To use no stochasticity and carry out deterministic modelling, we set this to `FALSE` (default). This vector refers to the affected transition stages in the same way as the Niche values, 3.2.4.

Possible arguments: "all" (all nonzero transition probabilities) or a vector of affected stages.

```
> transition_affected_env_mine <- "all"
```

```
[1] "all"
```

3.2.6 Stages affected by demographic stochasticity

Possible arguments: either "all" (all nonzero transition probabilities) *Demographic stochasticity is currently unimplemented.* or a vector of affected stages.

```
> transition_affected_demogr_mine <- "all"
```

```
[1] "all"
```

3.2.7 Type of environmental stochasticity

Do you want the environmental stochasticity to be normal or lognormal? Indicate "normal" or "lognormal" here. The default is a "normal" distribution.

```
> env_stochas_type_mine <- "normal"
```

```
[1] "normal"
```

3.2.8 Coefficients of variation in vital rates

Coefficients of variation in vital rates which are used to set how variable the environmental and demographic stocasticity should be. In this case they are set as 0.01 for all matrix elements.

```
> matrices_var_mine   <-
+  matrix(0.01, ncol = 1, nrow = nrow(matrices_mine), dimnames = list(NULL, "sd"))
```

A note about stochasticity: `demoniche` tests the matrix columns after stochasticity has been applied, to test if columns add to more than 1. If they sum to more than one, they are reduced to 1. The first row is disregarded, which corresponds for example to the number of seeds produced. This could lead to matrix columns that add to more than one if there is no seed stage.

3.2.9 Initial proportion

This is the initial proportion of individuals in all stages, and should be a vector of the same length as the number of stages, and should add to 1.

```
> proportion_initial_mine <- c(0.9818098089, 0.0006907668, 0.0069076675,
+                              0.0036840893, 0.0057563896, 0.0011512779)
```

```
[1] 0.9818098089 0.0006907668 0.0069076675 0.0036840893 0.0057563896
[6] 0.0011512779
```

3.2.10 Population density

Density of individuals (all stages, including seeds) in each grid cell. This number is automatically multiplied by the column named 'area Populations_mine, and should be in the same units as the area in the `Populations` object. This can be a single value, which is then used for all populations, or a vector if denisities in each population are known. This density value only affects the initial abundances, and not future projections.

```
> density_individuals_mine <- 20000
```

```
[1] 20000
```

3.2.11 Carrying Capacity (K)

The density dependence is set through two arguments, `K` and `Kweight`.

When populations are over the value in K (the number of individuals) they will reach carrying capacity, and will be reduced to the value of K. This takes place through a simple ceiling ("`ceiling`") functionality. The carrying capacity of newly colonized cells is set to the mean carrying capacity supplied.

If `K` is `NULL`, there is no carrying capacity limit in the simulations. In the demoniche model the K can be different for each (initial) population and each time period. If desired, `K` should correspond to either a vector of values with the same length as the number of populations, or a vector with the same length as the number of time periods (K for each timeperiod - the Niche values at each time period). It can also vary both for each population and each timeperiod, and should the be a matrix.

The `Kweight` argument defines which population stages should be affected by the density dependence. It should be a vector with the same length as the number of life stages. When calculating density dependent growth, the vector is multiplied by the population size when the population size is computed. A value of '1' means that the stage is counted as is, and a value above, means that it is counted proportionately more. If we want a stage to be ignored when determining density dependence, we make it a zero. In this case, we want mostly juveniles to be counted, but not seeds. The default is `FALSE` (same as a vector of ones). Beware, changing the `Kweight` means that populations could grow above or below the actual K set.

The values of carrying capacity could be defined by various means, for example from multiplying the population density by the Nichevalues, but this has to be done separately, outside of demoniche.

```
> K_mine              <- 10000
```

```
[1] 10000
```

```
> Kweight_mine <- c(0,1.5,1,1,1,1)
```

```
[1] 0.0 1.5 1.0 1.0 1.0 1.0
```

3.2.12 Scenario probabilities

With which probability should the scenario matrices should be drawn? Should be a vector of two numbers, and the default is equal probability.

```
> prob_scenario_mine <- c(0.5, 0.5)
```

```
[1] 0.5 0.5
```

3.2.13 Noise

This is currently not supported. A noise value of 1 is completely random - white noise (default), whereas a higher or lower value witll change it to a positive or negative temporal autocorrelation. This is option of specifying the intensity of temporal autocorrelation in the matrix selections, so that the probability of choosing the same matrix in subsequent years is decreased or increased. Is the change in probability that the same matrix as the last iteration will be chosen the next time period.

```
> noise_mine <- 0.1
```

```
[1] 0.1
```

3.2.14 Short distance dispersal

Dispersal between populations can be either short (with dispersal to the eight contiguous grid cells) or long-distance dispersal. Fraction of seeds that disperse short distance, beyond patch borders. Dispersal takes place to the 8 contiguous cells, with a proportion of 0.2 to cells contigous cells, and 0.5 to diagonal cells. The seeds that disperse to these cells will be subjected to the same matrix multiplication as all cells. If this is zero, no SDD is modelled.

```
> fraction_SDD_mine <- 0.05
```

```
[1] 0.05
```

3.2.15 Long distance dispersal

The long-distance dispersal is modelled by an exponential dispersal kernel with user-defined constants according to the formula $P = a * exp(-D^{(c/b)})$, with D = distance in kilometers between populations. These constants describe the shape of the species dispersal probability curve, and define a maximum distance beyond which dispersal is impossible. Long- and short-distance dispersal can be applied separately or together; alternatively, simulations without dispersal can be run.

The direction of seeds is deterined by the distances between populations. If this is zero, no LDD is modelled.

```
> fraction_LDD_mine <- 0.05
```

```
[1] 0.05
```

3.2.16 Dispersal kernel

Constants for LDD dispersal function as a vector in the format: c(a, b, c, Distmax) If the distance between populations is larger than Distmax (in the same units as the Populations, in this case in lat-long), there will be no dispersal between patches.

```
> dispersal_constants_mine <- c(0.7, 0.7, 0.1, 3)
```

```
[1] 0.7 0.7 0.1 3.0
```

3.3 Run `setup` function

Now we have defined all the information we need to run the setup function, and we can examine what objects we have in the work space. First we remove objects created earlier.

```
> rm(CC_nodispersal, eigen_results, Hmontana, hudsonia,hudvrs, niche_formulas, noCC_nodispersal)
```

```
NULL
```

```
> ls()

 [1] "K_mine"                          "Kweight_mine"
 [3] "Nichemap_mine"                   "Populations_mine"
 [5] "density_individuals_mine"        "dispersal_constants_mine"
 [7] "env_stochas_type_mine"           "fraction_LDD_mine"
 [9] "fraction_SDD_mine"               "matrices_mine"
[11] "matrices_var_mine"               "no_yrs_mine"
[13] "noise_mine"                      "prob_scenario_mine"
[15] "proportion_initial_mine"         "stages_mine"
[17] "sumweight_mine"                  "transition_affected_demogr_mine"
[19] "transition_affected_env_mine"    "transition_affected_niche_mine"
```

We specify these objects to each argument in the `demoniche_setup()` function and create the species object. Below is the complete function, with all the arguments specified. The setup function generates one species object with all the necessary information for the modeling in the workspace. It also checks the consistency of data, and prints any error messages. The order of the arguments does not matter, as long as all are present. The species object is also saved in the working directory, as an R-object (`.rda`) which can be loaded at a later time, or shared.

```
> demoniche_setup(modelname = "Hmontana",
+       Populations = Populations_mine, Nichemap = Nichemap_mine,
+       matrices = matrices_mine, matrices_var = matrices_var_mine, noise = noise_mine,
+       prob_scenario = prob_scenario_mine,
+       stages = stages_mine, proportion_initial = proportion_initial_mine,
+       density_individuals = density_individuals_mine,
+       fraction_LDD = 0.05, fraction_SDD = fraction_SDD_mine,
+       dispersal_constants = dispersal_constants_mine,
+       transition_affected_niche = transition_affected_niche_mine,
+       transition_affected_demogr = transition_affected_demogr_mine,
+       transition_affected_env = transition_affected_env_mine,
+       env_stochas_type = env_stochas_type_mine,
+       no_yrs = no_yrs_mine, K = K_mine, Kweight = Kweight_mine, sumweight = sumweight_mine)
```

We have now re-created the species object which is called the 'modelname' argument that we specified. As above, this can be used in the `demoniche_model` function (section 2.3).

Note that we defined the fraction of seeds dispersing long distances directly in the function, instead of `fraction_LDD_mine <- 0.05` and then defining the object in the setup function: `demoniche_setup(..., fraction_LDD = fraction_LDD_mine)`, we simply wrote the value directly: `demoniche_setup(..., fraction_LDD = 0.05)`. Either way works.

3.4 A minimal setup

As a minimal example, we now show the values that are needed to run a simulation with the default that are set by `demoniche`. A background grid that contains the populations is created, with only one time period.

```
> demoniche_setup(modelname = "Hmontana_minimal",
+       Populations = Populations_mine, matrices_var = matrices_var_mine,
+       matrices = matrices_mine,
+       stages = stages_mine,
+       proportion_initial = proportion_initial_mine,
+      density_individuals = density_individuals_mine,
+      no_yrs = no_yrs_mine)
> example_minimal <- demoniche_model(modelname = "Hmontana_minimal", Niche = FALSE,
```

```
+                          Dispersal = FALSE, repetitions = 1,
+                         foldername = "Hmontana_minimal")

[1] Starting projections for repetition: 1
[1] Projecting for scenario/matrix: Reference_matrix
[1] Projecting for scenario/matrix: Matrix_1
[1] Projecting for scenario/matrix: Matrix_2
[1] Projecting for scenario/matrix: Matrix_3
[1] Projecting for scenario/matrix: Matrix_4
[1] Calculating summary values
[1]     All repetitions completed!
```

4 Modifying demoniche functions

We encourage users to inspect the code in the demoniche functions, to find out how the algorithms are set up. By typing in only the function names in R (i.e. demoniche_setup without parentheses) directly into the console, the algorithm is printed.

If the user wishes to change how a function is carried out, this is possible and encouraged. Briefly, one could paste the code to a new R-script, make the desired changes to the function definition, and then read (or source) the new function into the workspace. Then the modified function can be used to run simulations, just as the original package function.

For example, say we wish to change the short-distance dispersal function. Currently propagules can spread to the 8 neighboring grid cells, as the function now is defined (the neighborhood index in the species object). In this case, whenever two patches are 1.0 or 1.4 degrees apart, we know that they are contiguous or diagonal from each other and propagules flow between them.

```
> Hmontana$neigh_index

[1] 1.0 1.4
```

The neighborhood index is defined in the demoniche_setup function (around line 100). First a distance matrix between all locations in the Niche values file is calculated, then the second and third largest values are saved in the species object. The behavior of this code can be examined with the original coordinates from the Niche values.

```
dist_latlong_try <- round(as.matrix(dist(Niche_ID[,2:3])), 1)
neigh_index_try <- sort(unique(as.numeric(dist_latlong)))[2:3]
```

We wish to modify this, so that migrating propagules only can spread to the 4 neighboring cells, directly contigous cells. We would have to change the function so that only the first value is saved in the neighborhood index, i.e. only 2 in this case (not 2:3), in the demoniche_setup function.

To make our own function, we first type: demoniche_setup into the console. The current function is returned to us, starting with > demoniche_setup function(modelname, Populations,...). We copy everything to a new R script, taking care to copy the last } which ends the function. We change the top of the function so that it becomes a standard function definition: demoniche_setup_fourneighbors <- function(modelname, Populations,..). Here we can also change the actual name of the function, to distinguish it from the original function, maybe: demoniche_setup_fourneighbors. We then change the lines in the function that we wish to modify: neigh_index <- sort(unique(as.numeric(dist_latlong)))[2] to only save the second number. If we run this function we have defined a new function called demoniche_setup_fourneighbors (check it!).

It is sometimes easier to save an R-script with only the function, like for example fcn_fourneighbors.R. Then the command source("fcn_fourneighbors.R") can be used which reads the function into the working space (just like when loading a package).

To run our modified function, we do the same thing as when running the original setup function but with the new function name: `demoniche_setup_fourneighbors(modelname = "Hmontana", Populations = Populations_mine,...)`. This will create a species object in which the `Hmontana$neigh_index` only has one number. When we run the model function on this object, dispersal will only take modelled to the four contigous cells.

The user might also want to change how the Niche values affect the transition matrices. This is located in the inner `demoniche_population` function, at around line 211, which is currently multiplicative:

```
one_mxs[transition_affected_niche] <- one_mxs[transition_affected_niche] * onepopulation_Niche
```

5 Trouble-shooting

`demoniche` has many parameters, and it might happen that there are errors. A good idea is to look at the error message returned by `R` and see where in the code this is taking place. This might give us clues to where there error is coming from. Also please ask questions in the mailing list on R-Forge.

6 Other resources

Related models: free-standing demographic/ hybrid modelling software with mostly unmodifiable source codes: Vortex (Lacy 1993), ULM (Legendre and Clobert 1995), RamasGIS/ Metapop (Akaçakaya and Root 2005), Prunus (Sebert-Cuvillier et al. 2009), BioMove (Midgely et al. 2010), Patch (Schumaker 1998, Schumaker et al. 2004), TreeMig (Lischke et al. 2006). Demographic modelling packages in R, limited to single populations: popbio (Stubben and Milligan 2007) Rramas (de la Cruz 2010) and demography (Hyndman 2010).

Related R-packages: `popbio`, `Rramas`, `demography`.

Recommended packages: `sp` to load shapefiles, `BIOMOD` and `dismo` to carry out Species Distribution Modelling, `SDMTools` to calculate fragmentation indices.

7 Acknowledgements

Many thanks to Rebecca Swab for greatly improving the user-friendliness of the model and manual, and patiently working through bugs. Many thanks to François Guilhaumon for generously sharing knowledge about coding and package-building in R. Many thanks to Regan Early for comments and advice about demographic modelling. Thanks to popbio and BIOMOD packages for inspiration.

8 References

Akçakaya H.R., Burgman M.A., Kindvall O., Wood C.C, Sjogren-Gulve P., Hatfield J.S. and McCarthy M.A. (Eds) 2002. Species conservation and Management: Case Studies. Oxford University Press.

Akçakaya, H. R. and Root, W. T. 2005. RAMAS GIS : linking spatial data with population viability analysis, version 5.0. Setauket, NY: Applied Biomathematics.

Caswell, H. 1989. Matrix Population Models: Construction, Analysis, and Interpretation. Sinauer Associates, Sunderland, Massachusetts.

de la Cruz, M. 2010. Rramas: Matrix population models. R package version 0.1-0. http://CRAN.R-project.org/package=Rramas

Gross, K., Iii, J.R.L., Frost, C.C., Morris, W.F., 1998. Modeling Controlled Burning and Trampling Reduction for Conservation of Hudsonia montana. Conservation Biology 12: 1291-1301.

Rob J Hyndman with contributions from Heather Booth, Leonie Tickle and John Maindonald (2011). demography: Forecasting mortality, fertility, migration and population data. R package version 1.10. http://CRAN.R-project.org/package=demography

Keith, D.A., Akçakaya, H.R., Thuiller, W., Midgley, G.F., Pearson, R.G., Phillips, S.J., Regan, H.M., Araújo, M.B., Rebelo, T.G., 2008. Predicting extinction risks under climate change: coupling stochastic population models with dynamic bioclimatic habitat models. Biology Letters 4: 560-563.

Lacy, R. C. 1993. VORTEX: a computer simulation model for population viability analysis. Wildlife Research 20: 45-65.

Legendre S. and Clobert J. 1995. ULM, a software for conservation and evolutionary biologists. Journal of Applied Statistics 22: 817-834.

Lischke, H., Zimmermann, N.E., Bolliger, J., Rickebusch, S. Loffler, T.J. 2006 TreeMig: a forest-landscape model for simulating spatio-temporal patterns from stand to landscape scale. Ecological Modelling 199: 409-420.

McCarthy, M. A. and Thompson, C. 2001. Expected minimum population size as a measure of threat. - Animal Conservation 4: 351-355.

Midgley, G.F., Davies, I.D., Albert, C.H., Altwegg, R., Hannah, L., Hughes, G.O., OHalloran, L.R., Seo, C., Thorne, J.H., Thuiller, W., 2010. BioMove - an integrated platform simulating the dynamic response of species to environmental change. Ecography 33: 612-616.

Morris, W.F., and D.F. Doak. 2002. Quantitative Conservation Biology. Theory and Practice of Population Viability Analysis. Sinauer Associates, Sunderland, Massachusetts.

Pagel, J., Schurr, F.M., 2011. Forecasting species ranges by statistical estimation of ecological niches and spatial population dynamics. Global Ecology and Biogeography. DOI: 10.1111/j.1466-8238.2011.00663.x

R Development Core Team (2010). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Schumaker, N.H. 1998. A Users Guide to the PATCH Model. EPA/600/R-98/135, U.S. Environmental Protection Agency, Environmental Research Laboratory, Corvallis, Oregon, 120.

Schumaker, N.Hm.T. Ernst, D. White, J. Baker, P. Haggerty, 2004. Projecting wildlife responses to Alternative Future Landscapes in Oregon's Willamette Basin. Ecological Applications, 14, 381-400.

Sebert-Cuvillier, E., Simonet, M., Simon-Goyheneche, V., Paccaut, F., Goubet, O., Decocq, G., 2009. PRUNUS: a spatially explicit demographic model to study plant invasions in stochastic, heterogeneous environments. Biological Invasions 12: 1183-1206.

Sjogren-Gulve, P. and T. Ebenhard. Eds. 2000. The Use of Population Viability Analyses in Conservation Planning. Ecological Bulletin 48: 9-21.

Stubben, C.J. and Milligan, B.G. 2007. Estimating and Analyzing Demographic Models Using the popbio Package in R. Journal of Statistical Software: 22-11.

Thuiller, W., Lafourcade, B., Engler, R., Araújo, M., 2009. BIOMOD - a platform for ensemble forecasting of species distributions. Ecography 32: 369-373.