```
l9  <- matrix(rep(1:9,9),9,9)
h30 <- helix()
#g54, last column is class label
g54 <- cbind(gauss54(),rep(1:5,e=15))
#iris data is a standard data-set
data(iris)
#check for duplicates else
#distance algorithms 'blow up'
iris.data <- unique(iris[,1:4])
iris.class <- iris[row.names(iris.data),5]
#some colours for the maps
classcol <- c("red","blue","green",
"black","yellow")
```

Bibliography

C. M. Bishop, M. S., and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998. URL citeseer.nj.nec.com/bishop98gtm.html. 6

M. Carreira-Perpinan. A review of dimension reduction techniques. Technical Report CS–96–09, Dept. of Computer Science, University of Sheffield, January 1997. 2

J. Edwards, K. Riley, and J. Eakins. A visual comparison of shape descriptors using multi-dimensional scaling. In *CAIP 2003, the 10th interantional conference on computer analysis of images and patterns*, pages 393–402, 2003. 2

J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, (53):325–328, 1966. 2

P. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985. 4

A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999a. URL citeseer.nj.nec.com/hyv99fast.html. 5

A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999b. 5

C. Jutten and J. Hérault. Blind separation of sources. *Signal Processing*, 24:1–10, 1991. 4

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 4

T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som pak: The self-organizing map program package, 1996. URL citeseer.nj.nec.com/kohonen96som.html. 4

J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 1-27(29):115–129, 1964. 3

I. Nabney. *NETLAB: Algorithms for Pattern Recognition*. Springer, 2001. 2, 6

K. Pearson. Principal components analysis. *London Edinburgh and Dublin Philosophical Magazine and Journal*, 6(2):559, 1901. 2

B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. 2, 3, 4

J. Sammon jnr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18: 401–409, 1969. 2, 5

L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4: 119–155, 2003. 6

D. F. Swayne, D. Cook, and A. Buja. XGobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998. URL citeseer.nj.nec.com/article/swayne98xgobi.html. 5

J. B. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(22):2319–2323, 2000. 6

M. Tipping and C. Bishop. Probabilistic principal component analysis. Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, 1997. 6

M. Tipping and D. Lowe. Shadow targets: a novel algorithm for topographic projections by radial basis functions. *Neurocomputing*, 19(1):211–222, 1998. 6

M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999. URL citeseer.nj.nec.com/tipping98mixtures.html. 6

J. Vesanto. SOM-based data visualization methods. *Intelligent-Data-Analysis*, 3:111–26, 1999. URL citeseer.nj.nec.com/392167.html. 4

A. Webb. *Statistical Pattern Recognition*. Wiley, 2002. ISBN 0470845139. 2

Jonathan Edwards & Paul Oman
Department of Informatics, University of Northumbria
Newcastle upon tyne, UK
{jonathan.edwards,paul.oman}@unn.ac.uk

R as a Simulation Platform in Ecological Modelling

Thomas Petzoldt

Introduction

In recent years, the R system has evolved to a mature environment for statistical data analysis, the development of new statistical techniques, and, together with an increasing number of books, papers and online documents, an impressing basis for learning, teaching and understanding statistical techniques and data analysis procedures.

Moreover, due to its powerful matrix-oriented language, the clear interfaces and the overwhelming amount of existing basic and advanced libraries, R became a major platform for the development of new data analysis software (Tierney, 2003).

Using R firstly for post-processing, statistical analysis and visualization of simulation model results and secondly as a pre-processing tool for externally running models the question arose, whether R can serve as a general simulation platform to implement and run ecological models of different types, namely differential equation and individual-based models.

From the perspective of an empirical ecologist, the suitability of a simulation platform is often judged according to the following properties:

1. learning curve and model development time,
2. execution speed,
3. readability of the resulting code,
4. applicability to one special model type only or to a wide variety of simulation methods,
5. availability of libraries, which support model development, visualization and analysis of simulation results,
6. availability of interfaces to external code and external data,
7. portability and licensing issues.

While questions 5 to 7 can be easily answered in a positive sense for the R system, e.g. portability, free GNU Public License, the availability of external interfaces for C, Fortran and other languages and numerous methods to read and write external data, the remaining questions can be answered only with some practical experience. As a contribution, I present some illustrative examples on how ecological models can be implemented within R and how they perform. The documented code, which can be easily adapted to different situations will offer the reader the possibility to approach their own questions.

Examples

The examples were selected to show different types of ecological models and possible ways of implementation within R. Although realism and complexity are limited due to the intention to give the full source code here, they should be sufficient to demonstrate general principles and to serve as an onset for further work[1].

Differential equations

The implementation of first-order ordinary differential equation models (ODEs) is straightforward and can be done either with an integration algorithm written in pure R, for example the classical Runge-Kutta 4th order method, or using the `lsoda` algorithm (Hindmarsh, 1983; Petzold, 1983), which thanks to Woodrow Setzer are both available in the **odesolve**-library. Compared to other simulation packages this assortment is still relatively limited but should be sufficient for Lotka-Volterra type and other small and medium scale ecological simulations.

As an example I present the implementation of a recently published Lotka-Volterra-type model (Blasius et al., 1999; Blasius and Stone, 2000), the constant period – chaotic amplitude (UPCA) model. The model consists of three equations for resource u, herbivorous v, and carnivorous w animals:

$$\frac{du}{dt} = au - \alpha_1 f_1(u,v) \tag{1}$$

$$\frac{dv}{dt} = -bv + \alpha_1 f_1(u,v) - \alpha_2 f_2(v,w) \tag{2}$$

$$\frac{dw}{dt} = -c(w - w^*) + \alpha_2 f_2(v,w) \tag{3}$$

where f_1, f_2 represent either the Lotka-Volterra term $f_i(x,y) = xy$ or the Holling type II term $f_i(x,y) = xy/(1+k_i x)$ and w^* is an important stabilizing minimum predator level when the prey population is rare.

To run the model as an initial value simulation we must provide R with (1) the model written as an R-function, (2) a parameter vector, (3) initial (start) values, and (4) an integration algorithm.

[1] Supplementary models and information are available on http://www.tu-dresden.de/fghhihb/petzoldt/modlim/

After loading the required libraries, the model equations (the Holling equation `f` and the derivatives `model`), can be written very similar to the mathematical notation. To make the code more readable, the state vector `xx` is copied to named state variables and the parameters are extracted from the parameter vector `parms` using the `with`-environment. The results of the derivative function are returned as list.

```
library(odesolve)
library(scatterplot3d)

f <- function(x, y, k){x*y / (1+k*x)} #Holling II

model <- function(t, xx, parms) {
    u      <- xx[1]
    v      <- xx[2]
    w      <- xx[3]
    with(as.list(parms),{
    du <-  a * u  - alpha1 * f(u, v, k1)
    dv <- -b * v           + alpha1 * f(u, v, k1) +
                           - alpha2 * f(v, w, k2)
    dw <- -c * (w - wstar) + alpha2 * f(v, w, k2)
    list(c(du, dv, dw))
  })
}
```

As a next step, three vectors have to be defined: the `times` vector for which an output value is requested, the parameter vector (`parms`), and the start values of the state variables (`xstart`) where the names within the vectors correspond to the names used within the `model` function.

```
times  <- seq(0, 200, 0.1)
parms  <- c(a=1, b=1, c=10,
            alpha1=0.2, alpha2=1,
            k1=0.05, k2=0, wstar=0.006)
xstart <- c(u=10, v=5, w=0.1)
```

Now the simulation can be run using either `rk4` or `lsoda`:

```
out <- as.data.frame(lsoda(xstart, times,
                           model, parms))
```

This should take only two seconds on a standard computer and finally we can plot the simulation results as time series or state trajectory (fig. 1):

```
par(mfrow=c(2,2))
plot(times, out$u, type="l", col="green")
lines(times, out$v, type="l", col="blue")
plot(times, out$w, type="l", col="red")
plot(out$w[-1], out$w[-length(out$w)], type="l")
scatterplot3d(out$u, out$v, out$w, type="l")
```

Of course, similar results can be obtained with any other simulation package. However, the great advantage using a matrix oriented language like R or Octave[2] is, that the core model can be easily integrated within additional simulation procedures. As an example a bifurcation (Feigenbaum)-diagram of the chaotic amplitude of the predator population can be computed. First a small helper function which identifies the peaks (maxima and minima) of the time series is needed. Within a vectorized environment this can be implemented very easily by selecting all those values which are greater (resp. lower for minima) as their immediate left and right neighbours:

```
peaks <- function(x) {
  l <- length(x)
  xm1 <- c(x[-1], x[l])
  xp1 <- c(x[1], x[-l])
  x[x > xm1 & x > xp1 | x < xm1 & x < xp1]
}
```

As a next step the integration procedure is included within a main loop which increments the variable parameter `b`, evaluates the peaks and updates the plot:

```
plot(0,0, xlim=c(0,2), ylim=c(0,1.5),
          type="n", xlab="b", ylab="w")
for (b in seq(0.02,1.8,0.02)) {
  parms["b"] <- b
  out <- as.data.frame(lsoda(xstart, times,
                       model, parms, hmax=0.1))

  l <- length(out$w) %/% 3
  out <- out[(2*l):(3*l),]
  p <- peaks(out$w)
  l <- length(out$w)
  xstart <- c(u=out$u[l], v=out$v[l], w=out$w[l])
  points(rep(b, length(p)), p, pch=".")
}
```

After a stabilization phase only the last third of the detected peaks are plotted to show the behavior "at the end" of the time series. The resulting bifurcation diagram (fig. 2) shows the dependence of the amplitude of the predator w in dependence of the prey parameter b which can be interpreted as emigration parameter or as predator independent mortality. The interpretation of this diagram is explained in detail by Blasius and Stone (2000), but from the technical viewpoint this simulation shows, that within the matrix-oriented language R the integration of a simulation model within an analysis environment can be done with a very small effort of additional code. Depending on the resolution of the bifurcation diagram the cpu time seems to be relatively high (several minutes up to one hour) but considering that in the past such computations were often done on supercomputers, it should be acceptable on a personal computer.

[2] http://www.octave.org

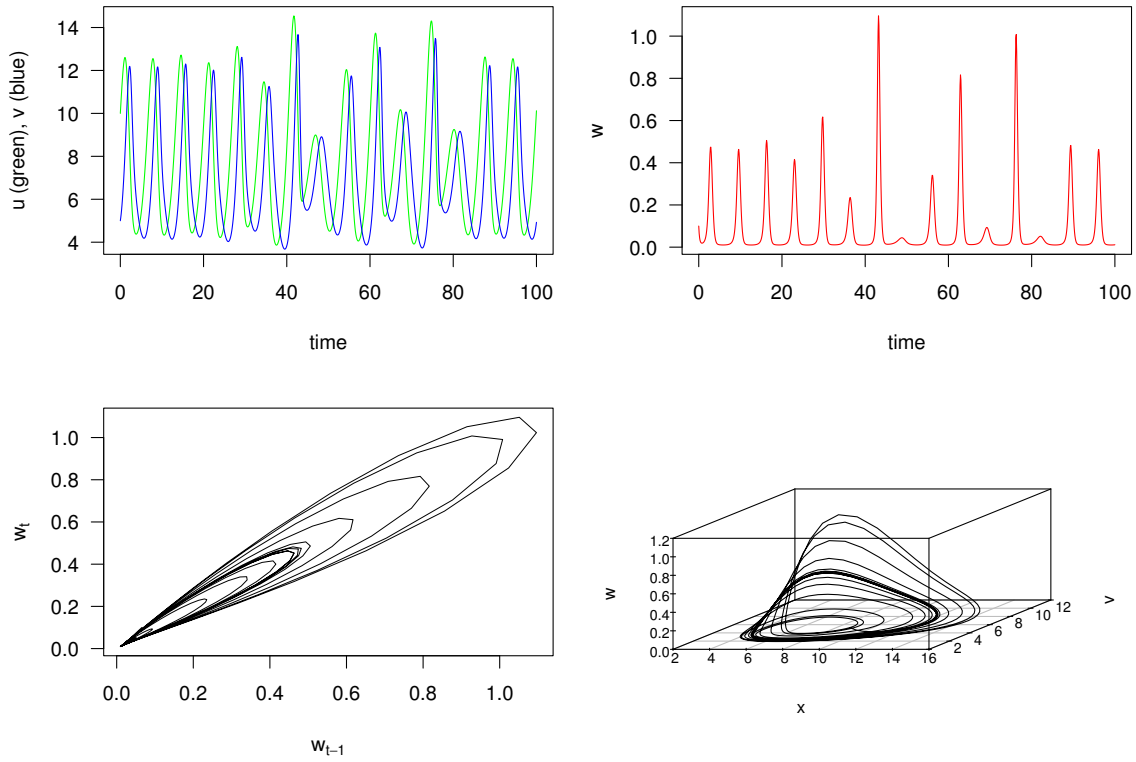

Figure 1: Simulation of an UPCA model, top left: resource (green) and prey (blue), top right: predator, bottom left: predator with lagged coordinates, bottom right: three dimensional state trajectory showing the chaotic attractor.

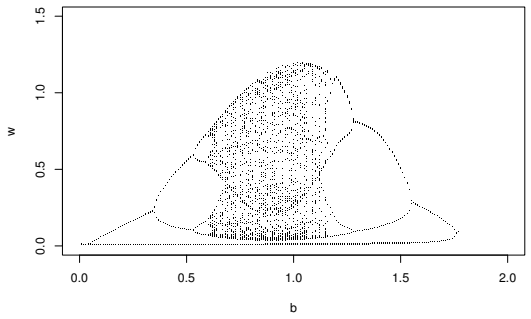

Figure 2: Bifurcation diagram for the predator w in dependence of prey parameter b.

Individual-based models

In contrast to ODE models, which in most cases work on an aggregated population level (abundance, biomass, concentration), individual based models are used to derive population parameters from the behavior of single individuals, which are commonly a stochastic sample of a given population. During the last decade this technique, which is in fact a creative collection of discrete event models, became widely used in theoretical and applied ecology (DeAngelis and Gross, 1992, and many others). Among them are very different approaches, which are spatially aggregated, spatially discrete (grid-based or cellular automata), or with a continuous space representation (particle diffusion type models).

Up to now there are different simulation tools available, which are mostly applicable to a relatively small class of individual-based approaches, e.g. SARCASim[3] and EcoBeaker[4] for cellular automata or simulation frameworks like OSIRIS (Mooij and Boersma, 1996). However, because the ecological creativity is higher than the functionality of some of these tools, a large proportion of individual-based models is implemented using general-purpose languages like C++ or Delphi, which in most cases, requires the programmer to take care of data input and output, random number generators and graphical visualization routines.

A possible solution of this tradeoff are matrix oriented languages like Matlab[5] (for individual-based models see Roughgarden, 1998), Octave or the S language. Within the R system a large collection of fundamental statistical, graphical and data manipu-

[3] http://www.collidoscope.com/ca/
[4] http://www.ecobeaker.com/
[5] http://www.mathworks.com/

lation functions is readily available and therefore it is not very difficult to implement individual-based models.

Individual-based population dynamics

The population dynamics of a *Daphnia* (water flee) population is shown here as an example. *Daphnia* plays an important role for the water quality of lakes (e.g. Benndorf et al., 2001), can be held very easily in the laboratory and the transparent skin makes it possible to observe important life functions using the microscope (e.g. food in the stomach or egg numbers in the brood pouch), so the genus *Daphnia* became an outstanding model organism in limnology, comparable to *Drosophila* in genetic research.

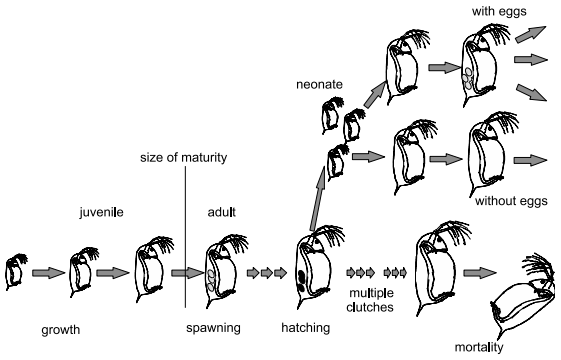

Figure 3: Parthenogenetic life cycle of *Daphnia*.

Under normal circumstances the life cycle of *Daphnia* is asexual (parthenogenetic) and begins with one new-born (neonate) female which grows up to the size of maturity (fig. 3). Then a number of asexual eggs are deposited into the brood pouch (spawning) which then grow up to neonates and which are released after a temperature-dependent time span into the free water (hatching). There are several approaches to model this life cycle in the literature (e.g. Kooijman, 1986; Gurney et al., 1990; Rinke and Petzoldt, 2003). Although the following is an absolutely elementary one which neglects food availability and simplifies size-dependent processes, it should be sufficient to demonstrate the principal methodology.

For the model we take the following assumptions (specified values taken from Hülsmann and Weiler (2000) and Hülsmann (2000)):

- The egg depelopment time is a function of water temperature according to Bottrell et al. (1976).
- Neonates have a size of 510 μm and the size of maturity is assumed to be 1250 μm.
- Juvenile length growth is linear with a growth rate of 83 $\mu m\, d^{-1}$ and due to the reproductive expense the adult growth rate is reduced to 80% of the juvenile value.
- The clutch size of each individual is taken randomly from an observed probability distribution of the population in Bautzen Reservoir (Germany) during spring 1999.
- The mortality rate is set to a fixed size-independent value of 0.02 d^{-1} and the maximum age is set to a fixed value of 30 d.

The implementation consists of six parts. After defining some constants, the vector with the cumulative distribution of the observed egg-frequencies, the simulation control parameters (1) and necessary helper functions, namely a function for an inverse sampling of empiric distributions (2), the life functions (3) of *Daphnia*: `grow`, `hatch`, `spawn` and `die` are implemented. Each individual is represented as one row of a data frame `inds` where the life functions either update single values of the individuals or add rows via `rbind` for each newborn individual (in `hatch`) or delete rows via `subset` for dead individuals. With the exception of the `hatch` function all procedures are possible as vectorized operations without the need of loops.

Now the population data frame is created (4) with some start individuals either as a fixed start population (as in the example) or generated randomly. The life loop (5) calls each life function for every time step and, as the `inds` data frame contains only one actual state, collects the desired outputs during the simulation, which are analysed graphically or numerically immediately after the simulation (6, see fig. 4). The example graphics show selected population statistics: the time series of abundance with an approximately exponential population growth, the mean length of the individuals as a function of time (indicating the presence of synchronized cohorts), the frequency distribution of egg numbers of adult individuals and a boxplot of the age distribution.

Additionally or as an alternative it is possible to save logfiles or snapshot graphics to disk and to analyse and visualise them with external programs.

```
#=============================================
# (1) global parameters
#=============================================
# cumulative egg frequency distribution
eggfreq     <- c(0.39, 0.49, 0.61, 0.74,
                 0.86, 0.95, 0.99, 1.00)
son         <- 510           # um
primipara   <- 1250          # um
juvgrowth   <- 83            # um/d
adgrowth    <- 0.8*juvgrowth # um/d
mort        <- 0.02          # 1/d
temp        <- 15            # deg C
timestep    <- 1             # d
steps       <- 40
#
# template for one individual
newdaphnia <- data.frame(age = 0,
                         size = son,
```

```
                      eggs = 0,
                     eggage = 0)
#=============================================
# (2) helper functions
#=============================================
botrell <- function(temp) {
    exp(3.3956 + 0.2193 *
       log(temp)-0.3414 * log(temp)^2)
}

# inverse sampling from an empiric distribution
clutchsize <- function(nn) {
    approx(c(0,eggfreq), 0:(length(eggfreq)),
           runif(nn), method="constant", f=0)$y
}
#=============================================
# (3) life methods of the individuals
#=============================================
grow <- function(inds){
    ninds       <- length(inds$age)
    inds$age    <- inds$age + timestep
    inds$eggage <- ifelse(inds$size > primipara,
                   inds$eggage + timestep, 0)
    inds$size   <- inds$size + timestep *
                   ifelse(inds$size < primipara,
                   juvgrowth, adgrowth)
    inds
}
die  <- function(inds) {
    subset(inds,
      runif(inds$age) > mort & inds$age <= 30)
}
spawn <- function(inds) {
  # individuals with size > primipara
  # and eggage==0 can spawn
  ninds          <- length(inds$age)
  neweggs        <- clutchsize(ninds)
  inds$eggs      <- ifelse(inds$size > primipara
                           & inds$eggage==0,
                           neweggs, inds$eggs)
  inds
}
hatch <- function(inds) {
    # individuals with eggs
    # and eggage > egg development time hatch
    ninds          <- length(inds$age)
    newinds        <- NULL
    edt <- botrell(temp)
    for (i in 1: ninds) {
        if (inds$eggage[i] > edt) {
          if (inds$eggs[i] > 0) {
            for (j in 1:inds$eggs[i]) {
                newinds <- rbind(newinds,
                             newdaphnia)
             }
          }
          inds$eggs[i]   <- 0
          inds$eggage[i] <- 0
        }
    }
    rbind(inds, newinds)
}
#=============================================
# (4) start individuals
#=============================================
inds <- data.frame(age  = c(7, 14, 1, 11, 8,
                            27, 2, 20, 7, 20),
                   size = c(1091, 1339, 618, 1286,
                            1153, 1557, 668, 1423,
                            1113, 1422),
                   eggs = c(0, 0, 0, 5, 0,
                            3, 0, 3, 0, 1),
                 eggage = c(0, 0, 0, 1.6, 0,
                            1.5, 0, 1.7, 0, 1.2))
#=============================================
# (5) life loop
#=============================================
sample.n        <- NULL
sample.size     <- NULL
sample.eggs     <- NULL
sample.agedist  <- NULL

for (k in 1:steps) {
    print(paste("timestep",k))
    inds <- grow(inds)
    inds <- die(inds)
    inds <- hatch(inds)
    inds <- spawn(inds)
    sample.n        <- c(sample.n,
                         length(inds$age))
    sample.size     <- c(sample.size,
                         mean(inds$size))
    sample.eggs     <- c(sample.eggs,
                         inds$eggs[inds$size >
                         primipara])
    sample.agedist  <- c(sample.agedist,
                         list(inds$age))
}
#=============================================
# (6) results and graphics
#=============================================
par(mfrow=c(2,2))
plot(sample.n, xlab="time (d)",
     ylab="abundance", type="l")
plot(sample.size, xlab="time (d)",
     ylab="mean body length (ţm)", type="l")
hist(sample.eggs, freq=FALSE, breaks=0:10,
     right=FALSE, ylab="rel. freq.",
     xlab="egg number", main="")
time <- seq(1,steps,2)
boxplot(sample.agedist[time],
        names=as.character(time), xlab="time (d)",
        ylab="age distribution (d)")
```

Particle diffusion models

In individual based modelling two different approaches are used to represent spatial movement: cellular automata and continuous coordinates (diffusion models). Whereas in the first case movement is realized by a state-transition of cells depending on their neighbourhood, in diffusion models the individuals are considered as particles with x and y coordinates which are updated according to a correlated or a random walk. The movement rules use polar coordinates with angle (α) and distance (r) which can be converted to cartesian coordinates using complex