Objects, Clones and Collections

Dynamic (Ecological) Models and Scenarios with simecol

Thomas Petzoldt
What are Ecological Models?

Two Examples from UseR!2006

Differential Equations (e.g. Lotka-Volterra)

\[
\frac{dX_1}{dt} = b \cdot X_1 - e \cdot X_1 \cdot X_2 \\
\frac{dX_2}{dt} = d \cdot X_1 + e \cdot X_1 \cdot X_2
\]

Individual-Based (Cellular automata, Random walk, ...)

\( X = \text{state.of(cell)} \)
\( N = \text{neighbours(cell)} \)

if \( X = 1 \) and \( N \in \{2,3\} \) then \( X := 1 \)

else if \( X = 0 \) and \( N = 3 \) then \( X := 1 \)

else \( X = 0 \)
Why Ecological Models in R?

R is more productive than other environments

1. More flexible than mouse-click environments,
2. Matrix orientation, Packages, Graphics, Sweave ...
3. More interactive than C++,
4. Modern computers are fast enough.
Why Ecological Models in R?

R is more productive than other environments

1. More flexible than mouse-click environments,
2. Matrix orientation, Packages, Graphics, Sweave ...
3. More interactive than C++,
4. Modern computers are fast enough.

R’s Soft Skills

▶ Scientific development model (books and publications),
▶ GPL makes sharing code easy,
▶ Enthusiastic useR! community.
Ecological Models in R?

Problems:

- 3 Modellers – 7 Programming styles,
- Copy & Paste to derive variants and scenarios?
- Reading code? ⇒ Better write a new program.
Ecological Models in R?

Problems:

- 3 Modellers – 7 Programming styles,
- **Copy & Paste** to derive variants and scenarios?
- **Reading code?** ⇒ Better write a new program.

Approach:

- Provide a standard approach,
- Use Object Oriented Programming – OOP,
- Package **simecol**: a model of ecological models.
Ecological Models in R?

Problems:

- 3 Modellers – 7 Programming styles,
- **Copy & Paste** to derive variants and scenarios?
- **Reading code?** ⇒ Better write a new program.

Approach:

- Provide a standard approach,
- Use Object Oriented Programming – OOP,
- Package **simecol**: a model of ecological models.

?? What do ecological / dynamic models have in common?
State Transition Diagram

\[ f(x, u, p, t) \rightarrow x(t) \rightarrow g(x, u, p, t) \rightarrow y(t) \]

- \( x(t) \): state transition function
- \( f(x, u, p, t) \): state transition function
- \( g(x, u, p, t) \): observer function
- \( u(t) \): inputs (external forcing)
- \( x(t_0) \): initial values
- \( t = t + dt \): time steps
- \( y(t) \): outputs
- \( t = t + dt \)
Specification
A whole model in one compact object.

S4 Classes: \texttt{odeModel}, \texttt{gridModel}, \texttt{rwalkModel}, \texttt{indbasedModel}

Equations and algorithms

Necessary data
iteration, user defined or imported from: \texttt{deSolve}, \texttt{rootSolve} or \texttt{ddeSolve}

data storage, screen output, logfiles, animation

The results

Tasks during initialization

Slots for: \texttt{Data} \hspace{1cm} \texttt{Functions}
Example: Stochastic Cellular Automaton

Spread of plants, animals, diseases, ... 

Survival rules

- Number of direct neighbours: 
  \( n_n = n_{\text{neighbours}} \in \{0 \ldots 8\}, \)
- Probability of seedling per adult neighbour: \( p_{\text{seed}} \),
- Total probability of seedlings per empty cell:
  \[ p_{\text{gen}} = 1 - (1 - p_{\text{birth}})^{n_n} \]
- Probability of death \( p_{\text{death}} \),
- Time step \( \Delta t = 1 \),
- State:
  - living cells: \( Z_{i,j} = t \) (age),
  - dead cells: \( Z_{i,j} = 0 \).
Stochastic Cellular Automaton in simecol

CA <- new('gridModel',
    main = function(time, init, parms) {
        Z <- init
        with(parms, {
            nn <- eightneighbours(Z >= adult) # direct neighbours
            pgen <- 1 - (1 - pseed)^nn # probability product
            zgen <- ifelse(Z == 0 & runif(Z) < pgen, 1, 0)
            zsurv <- ifelse(Z >= 1 & runif(Z) < (1 - pdeath), Z + 1, 0)
            zgen + zsurv
        })
    },
    parms = list(adult = 2, pseed = 0.2, pdeath = 0.1),
    times = c(from = 1, to = 50, by = 1),
    init = matrix(0, nrow = 80, ncol = 80),
    solver = 'iteration',
    initfunc = function(obj) { # obj = 'object in creation'
        init(obj)[38:42, 38:42] <- 1 # ... modify it
        obj # return the final object.
    }
)
Simulate the Model

library('simecol')
# (1) define or load the model
# source(); data(); ...

# (2) simulate the model
# Note: pass-back modification
CA <- sim(CA)

# (3) plot the model
# ... calls a specific plot-method
plot(CA)

# (4) Extract outputs
o <- out(CA)
Cloning ... 

Typical for Prototype-based-OOP (object-based, classless)

- Object creation: ex-nihilo or cloning and modification.
- Cloning: Creation time sharing (simple copy),
- Delegation: Run-time sharing
  (more memory efficient, but can confuse ecologists)
Cloning ...

Typical for Prototype-based-OOP (object-based, classless)

- Object creation: ex-nihilo or cloning and modification.
- Cloning: Creation time sharing (simple copy),
- Delegation: Run-time sharing
  (more memory efficient, but can confuse ecologists)

S4-classes not so far away from prototypes

- Cloning with assignment operator
  `scenario1 <- scenario2 <- CA`
  ⇒ Independent copies of the whole model object,
- Modify slots with replacement functions
  `parms, times, initfunc, . . . , equations, . . .`
Creating Scenarios

# Cloning
sc0 <- sc1 <- sc2 <- sc3 <- CA

# a series of scenarios with different settings
parms(sc1)$adult <- 1 # grow-up faster
parms(sc2)$pdeath <- 0.01 # live longer

# a scenario with random initialization
initfunc(sc3) <- <- function(obj) {
  init(obj) <- matrix(round(runif(80*80)), 80, 80)
  obj
}

# (2) simulate, plot ...
plot(sim(sc0))
plot(sim(sc1))
plot(sim(sc2))
plot(sim(sc3))
# Cloning

```r
sc0 <- CA  # keep CA as backup
parms(sc0)$pseed = 1  # <-- modify sc0 globally
sc1 <- sc2 <- sc3 <- sc0
```

# a series of scenarios with different settings

```r
parms(sc1)$adult <- 1
parms(sc2)$pdeath <- 0.01
```

# a scenario with random initialization

```r
initfunc(sc3) <- <- function(obj) {
  init(obj) <- matrix(round(runif(80*80)), 80, 80)
  obj
}
```

# ... simulate, plot ...

```r
plot(sim(sc0))
```
A Little Bit More Structure

The 'equations'-slot

▶ Modularization, functions, submodels,
▶ Implemented as list of functions in 'equations'
  (e.g. `neighb`, `generate`, `survive`)
▶ main-function should be as general as possible (call equations),
⇒ Possibility to derive scenarios with different functionality, e.g:

```r
# 8 direct neighbours (quick and simple)
equations(sc_8nb)$neighb = function(Z, adult, ...)
eightneighbours(Z >= adult)
```

```r
# neighborhood matrix (more general)
equations(sc_wdist)$neighb <- function(Z, adult, wdist)
neighbours(Z >= adult, wdist = wdist)
```
Variation of Model Structure
to compare models with different submodels

# A bell-shaped neighbourhood
x <- exp(-(seq(-2, 2, 0.5)^2))
parms(CA)$wdist <- outer(x, x)

# Settings for all scenarios
times(CA)['to'] <- 20
parms(CA)[c('pseed', 'pdeath')] <- c(0.9, 0.1)

# Cloning
sc_wdist <- sc_8nb <- CA

# Replace equation in scenario sc_wdist
equations(sc_wdist)$neighb <- function(Z, adult, wdist)
    neighbours(Z >= adult, wdist = wdist)

plot(sim(sc_8nb))
plot(sim(sc_wdist))
Observer Methods: The $g$ in the State Transition Diagram

Purpose

- Some models produce more data than required:
  - Condense data, perform statistical analysis.
- Simulations can be long and “anonymous”:
  - Output status info, write logfiles, show live animations.

How it works

- Observer is called once in each iteration step.
- Input = state – Output = data to store in ’out’
- Alternative observers are possible for one model, e.g. minimal observer for performance or elaborated observers for demo.
Aggregating Observers

aggregating_observer <- function(Z, time, ...) {
  loc <- which(Z > 0, arr.ind = TRUE)
  c(abundance = sum(Z > 0), age = mean(Z[Z > 0]),
     var.x = var(loc[,1]), var.y = var(loc[,2]))
}

# store, abundance, mean age and variances in x and y
observer(CA) <- aggregating_observer
CA <- sim(CA)
o <- out(CA)

with(o,
  plot(times, abundance)
  plot(times, age)
  plot(times, var.x, col='green')
  lines(times, var.y, col='red'))
}
Visual Observers

- Textual output of time and abundance and
- Animated histogram of age distribution during runtime.
Outlook

Package simecol

- Suitable for playing and for serious work as well.

Package simecolModels\(^1\)

- An open collection of (mostly published) models.
- Individual-based Simulation of Daphnia
  - Different versions in pure R or in R and C/C++,
  - Bioenergetic version (DEB), Cohort model (EBT),
  - Several 1000 individuals possible.
- Models for Marine Systems (among them 2D)
  - Use new differential equation solvers (package deSolve)
  - Thanks to Karline Soetaert\(^2\), NIOO, NL
- \ldots and more.

\(^1\)www.simecol.de, development on R-Forge
\(^2\)Note her upcoming book about practical ecological modelling in R.
One of Our Daphnia models

- Abundance = 7.27 1/L
- Sample size = 140
- Density vs. Length (mm)
- Density vs. Age (d)
- Phytoplankton vs. Day
- State Diagram

- Individual-based in R, bioenergetic core model in C/C++
- Live animation
A Horizontal 2D Zooplankton Model

Marine Zooplankton in the Scheldt

Zooplankton, mg/m³

- Partial differential equation (PDE) model,
- Available in simecolModels,
Random Walk Model in 3D

Live animation with RGL

- implemented as observer method

library(simecol); demo(rwalk3d)

▶ A live demo is possible – if you like . . .
Conclusions: Why using simecol?

Pre-defined structure:

- Compact, standardized representation of models,
- Improved readability (your code and that of your students).

Scenario control made easy:

- Formalized **Cloning** instead of cumbersome copy & paste,
- Standard methods for changing data and formula,
- No interference between different instances of one model.

Reproducible Science:

- Model input and outcome together in one object,
- Objects can be stored in binary or human readable form.
Conclusions: Why using simecol?

Pre-defined structure:
- Compact, standardized representation of models,
- Improved readability (your code and that of your students).

Scenario control made easy:
- Formalized Cloning instead of cumbersome copy & paste,
- Standard methods for changing data and formula,
- No interference between different instances of one model.

Reproducible Science:
- Model input and outcome together in one object,
- Objects can be stored in binary or human readable form.

⇒ simecol-Models: a nice gift for your friends.
Appendix
One simple example
Lotka-Volterra-Model

```r
lv <- new('odeModel',
  main = function (time, init, parms) {
    x <- init
    with(as.list(parms), {
      list(c(dx1, dx2))
    })
  },
  ## birth encounter death
  parms = c(b=0.2, e=0.2, d=0.2),
  times = seq(0, 100, 1),
  init = c(prey=0.5, predator=1)
)

plot(sim(lv))
```
Agent based models (ABMs) with R?
Data structures in R
Object oriented, high-level, performance-optimized

- vector
- matrix, data.frame
- list
- list of data.frames
- vector of lists
- etc ...

- “Large Blocks” ⇒ good Performance.
- Time critical code → C/C++, Fortran or JAVA.
Contact Information

Dr. Thomas Petzoldt
Technische Universität Dresden
Institut für Hydrobiologie
01062 Dresden
GERMANY

http://tu-dresden.de/Members/thomas.petzoldt

http://www.simecol.de

Clone sheep picture:

The examples were created with R 2.7.1, simecol 0.6 and simecolModels 0.2-3. Full source code of the examples is part of these packages.

To cite package 'simecol’ in publications, please use:

http://www.jstatsoft.org/v22/i09/.