PSPManalysis

A package for numerical analysis of physiologically structured population models

André M. de Roos

Institute for Biodiversity and Ecosystem Dynamics
University of Amsterdam

A.M.deRoos@uva.nl

Version January 5, 2018

Contents

Chapter 1

Preface

This software package is distributed to simplify the analysis of physiologically structured population models (PSPMs) or life history models in general. If you are not familiar with PSPMs there are many sources you can check, in particular the original book by Metz and Diekmann (1986), but a more gentle introduction is provided in De Roos (1997). An earlier version of this software has been used to produce the many bifurcation graphs of equilibria in structured population models that appear in De Roos and Persson (2013). The basic layer of the software has hence been tested quite extensively. The current version is built on top of that basic layer to make the implementation of a particular PSPM easier and to make the software package accessible from R or Matlab. The software can also be used from the command-line of any Unix-based system (Linux or Mac OS) without the overhead of R or Matlab. The entire software package `PSPManalysis`, which also includes a front-end for Matlab as well as Unix-command-line usage, can be found at my personal website and at Bitbucket. This vignette documents the R package version of `PSPManalysis`.

The package is free software and released under the GNU General Public License without any warranty or even the implied warranty of merchantability or fitness for a particular purpose (the official statement of the GPL). If you are using the software for publications, you are kindly asked to credit this software package by a reference to this documentation and the website that hosts the software package, as these are currently the only sources to be referred to.

In case you encounter any problem with the software package, please first verify the problem is not in your own model-specific file, but indeed is a bug in the general software layer. If you are convinced it is a bug in my programming, send me an email with as accurate a description of the problem as possible. Do not not forget to include your model-specific file and details about the invocation of the scripts that caused the problems. Any comments and feedback, both on the code and on the current manual is appreciated and will be considered carefully. In particular concrete comments, for example, explicit suggestions for textual changes in the manual and/or corrections of the mistakes (they are definitely there!) will be highly valued and acknowledged.

Development of this software was supported by funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement No. 322814

Chapter 2

Introduction

This software package implements numerical procedures for the analysis of physiologically structured population models (PSPMs). PSPMs represent a class of models that consistently translate continuous-time models of individual life history to the population level. The formulation of such models is discussed extensively in Metz and Diekmann (1986) and De Roos (1997) and is presented here only as far as needed for the use of the software.

The software allows for five different types of analyses of PSPMs:

- **Demographic analysis**: For linear PSPMs that do not account for density dependence or population feedback on the life history of individual organisms, the long-term population growth rate can be calculated. If the dynamics of such a linear PSPM would be simulated over time in the long run the population would grow exponentially or decline to zero with this population growth rate. The software also automatically calculates the sensitivity of this population growth rate with respect to all model parameters. Furthermore, the software calculates the stable population distribution, which characterizes the composition of the population during its exponential growth phase, and the reproductive value of the individuals in this stable population state as a function of their individual state.

- **Equilibrium analysis**: Equilibrium states can be computed for non-linear PSPMs that do account for density dependence or feedback of the population on the life history of individual organisms. These equilibrium states are computed as a function of a single model parameter, resulting in a parameterized curve of equilibrium states. Two types of special points can be detected on these equilibrium curves: limit points, also called saddle-node bifurcation points, and branching points or transcritical bifurcation points. Furthermore, the software allows for the computation of these two types of bifurcation points as a function of two model parameters.

- **Analysis of evolutionary fixed points**: During the computation of equilibrium curves of a non-linear PSPM the software also can check whether an evolutionary singular point as defined by Adaptive Dynamics or ESS-theory (Dieckmann 1997, Metz et al. (1996)) is encountered. These singular points are subsequently classified as either a convergent stable strategy (CSS), an evolutionary branching point (EBP) or an evolutionary repellor (ERP) (Geritz et al. 1998). The software can also compute the value of a detected evolutionary singular point as a function of a second model parameter and can, starting from a detected evolutionary singular point, compute the pairwise invasibility plot (Dieckmann 1997, Metz et al. (1996)).

- **Ecological dynamics simulation**: The ecological dynamics of PSPM can be computed using the *Escalator Boxcar Train* (De Roos 1988, De Roos, Diekmann, and Metz (1992)), a numerical method especially designed for numerical integration of the partial differential equations that are the mathematical representations of PSPMs. A separate software package, EBTtool, for

computing the ecological dynamics of PSPMs has been available already for many years. The EBTtool consists of a graphical user interface including extensive plotting capabilities and a computational engine. A trimmed down version of this computational engine is included in the PSPManalysis package.

- **Evolutionary dynamics simulation**: The dynamics of life history trait values, which in the model occur as parameters, can be simulated over evolutionary time scales, using the canonical equation for adaptive dynamics as explained in (Dieckmann and Law 1996). These evolutionary dynamic simulations are based on the assumption that the system approaches an ecological equilibrium in between mutation events, which change the value of the life history trait. The evolutionary rate of change is proportional to the selection gradient in the ecological equilibrium and the population birth rate.

The software package consists of a collection of routines implemented in `C` with front-ends that allows the software to be used from `R`. The implementation of the elements of the PSPM under study can be programmed in either `R` or `C` using the template files provided with the package. Implementation of the user-defined ingredients of the PSPM to be analyzed in `R` is easier and to most users probably more familiar, but the user should be aware that implementing the user-defined ingredients of the PSPM in `C` will decrease computation times by roughly 2 orders of magnitude. Specifying the user-defined model ingredients in `R` hence comes at the price of computations being excruciatingly slow. In many cases the added difficulty of using `C` will therefore pay off.

> Notice that the software package can also be used from `Matlab` as well as from the Unix command-line, but this requires that the version of the package available from my website is downloaded and installed. There is also an older (most likely outdated) version of software manual that explains the syntax of the commands to use the package from `Matlab` or the Unix command-line.

Irrespective of whether the model is implemented in `R` or in `C` to use the package a recent version of the *gcc* or *clang* compiler has to be installed, such that R commands such as

```
1 system("R CMD SHLIB PSPMequi.c")
```

can be succesfully executed by the various `R` functions in the package. On Linux systems it is quite standard that a *gcc* or *clang* compiler is installed by default. On Mac OS systems, these compilers can be installed by the following steps:

1. Launch the `Terminal` application, found in `/Applications/Utilities/`
2. Type the following command string:

   ```
   xcode-select -install
   ```

Windows users will have to install the Windows toolset from cran.r-project.org. Getting modules compiled by other compilers to communicate with R is much more complicated and therefore not recommended. More details can be found on cran.r-project.org. To use the package on Windows systems it is recommended to follow these steps:

1. Download `RStudio` from www.rstudio.com
2. Download and install a base installation of `R` for Windows from cran.r-project.org
3. Install and download the latest version of `Rtools.exe` from cran.r-project.org. This installs the program in `C:\Rtools`. During installation select the default installation. Also let the installer adjust the `PATH` value to include `C:\Rtools\bin` and `C:\Rtools\gcc-X.X.X\bin`

4. (This step is not needed when using R studio) Also add `C:\Program Files\R\R-3.X.X\bin` to the `PATH` variable

   Notice that working with `PSPManalysis` on a Windows network share (a filepath starting with `\\machine\...`) is not supported: such paths will need to be mapped to a network drive (e.g. `Z:\`) and used from this mapped drive letter. This is not an issue related to `PSPManalysis` itself, but is actually an issue of `R`.

The basic methodology to numerically compute the equilibrium of a PSPM has been presented in Kirkilionis et al. (2001) and Diekmann, Gyllenberg, and Metz (2003), while De Roos (2008) presented the modification of the latter approach to compute the demographic characteristics of a linear PSPM. For the interested reader this manual provides a brief sketch of this computational approach in chapter 10.

Chapter 3

Demographic analysis of linear PSPMs

3.1 Model formulation and ingredients

The core of a linear PSPM consists of a model of the individual life history that is based on the following assumptions:

- Individuals are characterized by their *individual* or *i-state*, which is a (finite) set of physiological characteristics (traits such as age, size, sex, energy reserves):

  $$\boldsymbol{\chi} = (\chi_1, \ldots, \chi_k) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

- Individuals are born with an *i-state* $\boldsymbol{\chi}_b$ that is one of a finite set of possible states at birth:

  $$\boldsymbol{\chi}_b \in \{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$$

  with each potential state at birth $\boldsymbol{\phi}_j$ a valid *i-state*:

  $$\boldsymbol{\phi}_j = (\phi_{j1}, \ldots, \phi_{jk}) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

- Development follows a deterministic process that is continuous in time:

  $$\frac{d\boldsymbol{\chi}}{da} \;=\; g(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$$

  The development rate $g(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$ is a function of the individual state and the individual's state at birth

- Reproduction is modeled by a per-capita offspring production rate (or fecundity) $\beta(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$, dependent on the individual state and the individual's state at birth

- Mortality is modeled by a per-capita death rate $\mu(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$, dependent on the individual state and the individual's state at birth

All assumptions above are characteristic for the general class of PSPMs. The most restrictive of these assumptions concerns the deterministic development process. Biologically, this assumption implies that all individuals that are born with the same state at birth will remain identical throughout their life and will hence not diverge in their *i-state* characteristics. Reproduction and mortality on the other hand are at an individual level considered as stochastic processes, which translate to per-capita rate functions at the population level, given that it is assumed that the number of individuals is large (technically speaking the number of individuals for every possible *i-state*).

3.2 An example model for demographic analysis

The steps needed for the implementation of a particular PSPM will be discussed using a simple model for the life history of the Mediterranean fruitfly, which is also discussed in De Roos (2008). The individual life history in this model is only age-dependent with both age-dependent birth and mortality rates. The PSPM for this model can be described by the following partial differential equation (PDE) for the population age-distribution $n(t, a)$:

$$\frac{\partial n}{\partial t} + \frac{\partial n}{\partial a} = -\mu(a)\, n(t,a)$$

$$n(t,0) = \int_{A_j}^{\infty} \beta(a)\, n(t,a)\, da$$

$$\beta(a) = \beta_0\, e^{-\beta_1 (a - A_j)}, \qquad \text{if } a > A_j$$

$$\mu(a) = \mu_0\, e^{\mu_1 a}$$

The first, partial differential equation above describes the changes in the population age-distribution $n(t, a)$ through aging ($\partial n/\partial a$) and mortality, which is modeled by the mortality rate $\mu(a)$. The second equation, representing the boundary condition for the partial differential equation, describes the total population reproduction rate $n(t, 0)$, which equals the cumulative fecundity of all individuals older than A_j, the age at maturation. The mortality rate $\mu(a)$ is an exponentially increasing function of age, whereas the fecundity $\beta(a)$ is highest for just maturing individuals ($a = A_j$) and decreases exponentially with age afterward.

As listed in section 3.1, individuals are assumed to be born with an *i-state* $\boldsymbol{\chi}_b$ that is one of a finite set of possible states-at-birth, each of which is a valid *i-state*:

$$\boldsymbol{\chi}_b \in \{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}, \qquad \boldsymbol{\phi}_j = (\phi_{j1}, \ldots, \phi_{jk}) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

Given that individual age is the only *i-state* variable in the Medfly model, all individuals have the same state at birth and hence $m = 1$. The option to specify multiple states-at-birth is hence not relevant for the example model discussed in this implementation chapter. This might hold more generally; most if not all physiologically structured population models that have been reported on in the literature so far are characterised by such a unique state-at-birth for all individuals. Nonetheless, the option to define multiple states-at-birth opens up some interesting research possibilities, which are discussed further in section 9.1.

Since models involving multiple states-at-birth are not very common, information that relates to this option will be distinguished in the text by setting them apart in paragraphs like this one. The index j will be used to refer to the index of a particular state-at-birth in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$. The number m of possible states-at-birth is set dynamically in the model file.

3.3 Implementation of the example model

3.3.1 Implementation of the model in `R`

The implementation of this model, which I will refer to as the Medfly model requires the specification of 3 constants and 3 functions describing the life history. The necessary pieces of `R`-code are discussed

in detail in the next subsections. The code can be found in the file `Medfly.R`, which can be opened by executing the command `showpspm("Medfly.R")` at the command line. To implement your own model it is advisable to use one of the example models, which can be listed using the utility function `showpspm()`, as a basis for the implementation. To do so, you can copy the contents of the file `Medfly.R` to a new file in `Rstudio`'s built-in editor and save this new file with a new name. The extension of your model-specific file should however remain `'.R'`.

3.3.1.1 Problem dimensions

The first variable to define, `PSPMdimensions`, is a vector with the named elements `PopulationNr`, `IStateDimension` and `LifeHistoryStages` that specify the dimensions of the model:

Code block 3.3.1.1

```
PSPMdimensions <- c(PopulationNr = 1, IStateDimension = 1, LifeHistoryStages = 2)
```

The software can simultaneously compute the population growth of more than a single population. The vector element `PopulationNr` of `PSPMdimensions` has to be defined equal to the number of structured populations accounted for in the model. For the Medfly example this is obviously equal to 1. The vector element `IStateDimension` of `PSPMdimensions` defines the dimension of the individual state. As only age characterizes the individuals in the Medfly model, this variable is defined equal to 1. Finally, the element `LifeHistoryStages` of `PSPMdimensions` has to be defined equal to the number of life stages that can be distinguished in the individual life history. While integrating the ODEs for the individual life history numerical problems may occur when the right hand side of the ODEs changes abruptly in value at a certain threshold value of the individual state, as a consequence of discontinuities in the development rate, the mortality rate or the fecundity. Each of such thresholds in the individual life history should be distinguished as a stage boundary. In the Medfly model the fecundity $\beta(a)$ changes from 0 just before $a = A_J$ to β_0 at $a = A_J$ and $\beta_0 \exp(-\beta_1(a - A_j))$ at larger ages. At $a = A_j$ $\beta(a)$ thus exhibits a discontinuity, which separates the juvenile and the adult stage from each other. The element `LifeHistoryStages` of `PSPMdimensions` is therefore set equal to 2.

3.3.1.2 Optional numerical settings

The next variable specified in the `Medfly.R` file is the vector `NumericalOptions`, which can contain a variable number of named vector elements:

Code block 3.3.1.2

```
NumericalOptions <- c(MIN_SURVIVAL  = 1.0E-9,   # Survival at which individual is considered dead
                      MAX_AGE       = 100000,   # Absolute maximum individual age
                      DYTOL         = 1.0E-7,   # Variable tolerance
                      RHSTOL        = 1.0E-8)   # Function tolerance
```

The specification of `NumericalOptions` is optional and can be left out, if default values are acceptable. A list of all possible vector elements that can be included in the `NumericalOptions` variable is provided in section 9.3.

The vector element `MIN_SURVIVAL` of `NumericalOptions` determines the threshold of the survival probability below which an individual is considered dead. The integration over the individual life history stops whenever the survival probability falls below this threshold value. In the code above the minimum survival is set to 10^{-9}, which is in fact the default value and is hence superfluous. Note that the value of `MIN_SURVIVAL` can *not* be set equal to 0. As an alternative to using 0 `MIN_SURVIVAL` can be set to a very small value like 10^{-100}.

The vector element `MAX_AGE` of `NumericalOptions` can be used as an alternative to determine the end of an individual life and to stop the integration over the individual life history. In the Medfly

model there is no maximum individual age and hence the variable is set to a very high value (100000), which the individuals will never reach, because before that age their survival probability has already dropped below its threshold value (10^{-9}).

The remaining two vector elements `DYTOL` and `RHSTOL` of `NumericalOptions` determine whether a solution has been found. In general, both demographic analysis as well as equilibrium analysis of PSPMs boils down to solving a system of nonlinear equations that can be represented as $G(y) = 0$ for a set of unknowns y in an iterative manner. The subsequent estimates of the solution in the Newton iterations can be labeled as y_p and y_{p+1}. A solution is now considered to be located if both of the following conditions hold:

$$\|y_{p+1} - y_p\| < \epsilon_y$$

$$\|G(y_{p+1})\| < \epsilon_G$$

where $\|.\|$ refers to the Euclidean norm. `DYTOL` and `RHSTOL` are the quantities ϵ_y and ϵ_G, respectively. Increasing (decreasing) their value leads to easier (harder) acceptance of a set of unknowns as a solution to the system of equations $G(y) = 0$. The definition of these two accuracies in the code box above is in fact superfluous as they are defined equal to their default values (see section 9.3).

3.3.1.3 Default parameters

The last variable to be defined is the vector `DefaultParameters`, which should contain named vector elements that specify the name and default value of all parameters in the model:

Code block 3.3.1.3

```
1 DefaultParameters <- c(Beta0 = 47.0, Beta1 = 0.04, Aj = 11.0, Mu0 = 0.00095, Mu1 = 0.0581)
```

The names of the vector elements (parameters) can be used in the programming of the life history functions of the model and are furthermore used to make the output files produced by the program more readable. These output files contain a small header text indicating among other details which parameter values were used for the computation of the results contained in the output file. In this report the parameter names are listed together with their value.

3.3.1.4 States at birth

The first function to be implemented for a particular life history model should be called `StateAtBirth()` and should define for every population in the model the actual value of the different individual state variables $\boldsymbol{\phi}_j = \left(\boldsymbol{\phi}_{j1}, \ldots, \boldsymbol{\phi}_{jk}\right)$ for every possible state-at-birth that an individual can be born with (i.e. the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$).

The function should return a vector with as many named vector elements as there are i-state variables. Each vector element should specify the name of the particular i-state variable and the numeric value with which the individual is born. The names of the vector elements can be used conveniently in the functions below that define the life history processes.

> If individuals can differ in their individual state at birth this function should return a matrix with the number of rows equal to the number of possible states at birth and the number of columns equal to the number of i-state variables. Each row then specifies the value of the individual state variable of the particular state at birth. In case the model accounts for multiple, structured populations this function should return a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables.

In case the model accounts for multiple, structured populations and individuals can differ in their individual state at birth this function should return a 3-dimensional array with the first dimension having a length equal to the number of structured populations in the problem, the second dimension equal to the number of possible states at birth and the third dimension equal to the number of i-state variables.

For the Medfly model age is the only individual state variable and its value at birth is of course 0:

Code block 3.3.1.4

```
StateAtBirth <- function(E, pars)
{
  with(as.list(c(pars)),{
        # We model a single structured population with a single i-state variable (age)
        c(Age = 0.0)
      })
}
```

Notice that the arguments of the function `StateAtBirth()` contain a vector `E` in addition to the vector with parameter values `pars`. The vector `E` will contain the values of the environment variables during equilibrium computations of PSPMs (see sections 4.1 to 4.4). In demographic analysis of PSPMs this variable is non-functional and is best ignored, using it in a statement inside the routine may even cause the program to crash. The only reason for the presence of this variable among the function arguments is to keep the function declaration the same for both demographic and equilibrium analysis computations. In principle, the same model-specific file can hence be used for both types of analysis. The variable `E` will for the same reasons also be part of the headers of the next 2 routines.

3.3.1.5 Boundaries between consecutive stages

The next function `LifeStageEndings()` determines the boundaries between consecutive stages in the individual life history. It should return a variable named `maturation`, the value of which specifies the threshold value at which the current life stage of the individual ends and the individual matures to the next life history stage. The life stage that the individual is in at the moment this routine is called, is determined by the function argument `lifestage`, which has a value of 1 if the individual is in the first life stage and a value equal to `PSPMdimensions["LifeHistoryStages"]` if it is in the last life stage. The end of the current life history stage, as indicated by `lifestage`, occurs when this threshold value becomes 0 and switches sign from negative to positive. For the end of the last life stage the death of old age, either by reaching the maximum age (`NumericalOptions["MAX_AGE"]`) or by reaching the minimum survival threshold (`NumericalOptions["MIN_SURVIVAL"]`), does not have to be specified separately, the program takes care of that automatically. For the final life stage hence return a constant negative value (for example, -1). In case the model accounts for multiple, structured populations the return variable `maturation` is a vector with the number of elements equal to the number of structured populations in the problem, while the argument `lifestage` is also a vector of a length equal to the number of structured populations in the model.

In the Medfly model only the end of the larval stage has to be specified, as shown below:

Code block 3.3.1.5

```
LifeStageEndings <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(E, pars, istate)),{
        maturation  = switch(lifestage, Age - Aj, -1)
      })
}
```

The threshold value returned to the program in `maturation` will in general depend on the individual state variables, possibly on the individual's state-at-birth and will be different for individuals in different life stages. For this reason, the function `LifeStageEndings()` has as arguments `lifestage`, specifying the life stage that the individual is currently in, `istate`, the individual state, and `birthstate`, the individual's state-at-birth in addition to the arguments `E` and `pars` that have the same interpretation as discussed above for the function `StateAtBirth()`.

This routine will be called as many times as there are possible states-at-birth, because the state-at-birth may influence the threshold between consecutive life stages. The same holds for the next 2 routines discussed, which define changes in the *i-state* variables, the fecundity and the mortality of individuals. In essence, individuals with different states-at-birth are treated as constituting subpopulations within the same structured population. Because of the possible dependence on the state-at-birth the variables `birthstate` and `BirthStateNr` are passed as arguments to the function `LifeStageEndings()` as well as to the 2 functions discussed below. These arguments contain the values of the *i-state* variables and the index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$, respectively, for which the routine is invoked and for which the threshold between consecutive life stages has to be evaluated.

3.3.1.6 Life history rates

The next function, named `LifeHistoryRates()`, specifies the life history rates of an individual. The function should return a list with 3 components, named `development`, `fecundity` and `mortality`. The components should have the following structure:

- `development`: This component of the returned list specifies the right-hand side of the ODE:

  $$\frac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$$

  determining the continuous development of the individual state variables during the life history. It should be a vector of length equal to the number of i-state variables. Each element specifies the rate of development for the particular i-state variable.

  Notice that the development rate may differ in different life stages, for example growth in body size may be different for juveniles and adults in case adults invest a lot of energy into reproduction. The development rates should then be specified dependent on the current life stage the individual is in, which is determined by the function argument `lifestage`. The development rate may furthermore depend on the individual state variables and on the parameters, i.e. the values of the arguments `istate` and `pars`, respectively, but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate` and `BirthStateNr`, respectively.

  In case the model accounts for multiple, structured populations this component should be a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables. In this case, the value of `development[p,i]` determines for each structured population `p` the development in the individual state variable `i`.

- `fecundity`: This component of the returned list specifies the current fecundity of the individual. In the most common case of a unique state-at-birth and a single structured population, like in the PNAS2002 model, the component `fecundity` should be a single value.

  The fecundity will certainly depend on the life stage that the individual is in (only adults reproduce), which is contained in the function argument `lifestage`, on the individual state

variables and on the parameters, i.e. the values of the arguments `istate` and `pars`, respectively, but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate` and `BirthStateNr`, respectively.

In case the model accounts for multiple, structured populations this component should be a matrix of fecundities with the number of rows equal to the number of structured populations in the problem and a single column. In case individuals can be born with different states at birth the component should have a number of columns equal to the number of states at birth. In this latter case not only the fecundity (i.e. the number of offspring produced per unit time) has to be specified, but also the state-at-birth of the produced offspring. Therefore, this function has to assign values to the matrix `fecundity[p,b]`, which determines for the population with index `p` the number of offspring produced per unit time with state-at-birth with index `b` in the set $\{\phi_1, \ldots, \phi_m\}$. Each column should hence specify the number of offspring produced with the particular state at birth.

- `mortality`: A single value specifying the current mortality rate that the individual experiences, possibly dependent on the life stage the individual is in at the moment this routine is called (given in the function argument `lifestage`), the current values of the individual state variables and parameters, i.e. the values of the arguments `istate` and `pars`, respectively, and the individual's state-at-birth (current values and index in the set $\{\phi_1, \ldots, \phi_m\}$ given by `birthstate` and `BirthStateNr`, respectively).

  In case the model accounts for multiple, structured populations this argument is a vector of mortality rates with the number of elements equal to the number of structured populations in the problem.

For the Medfly model the function `LifeHistoryRates()` is specified as follows:

Code block 3.3.1.6

```
LifeHistoryRates <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(pars, istate)),{
        list(
            # We model a single structured population (nrow=1) with a single i-state variable (age)
            development = 1.0,
            fecundity   = switch(lifestage, 0, Beta0*exp(-Beta1*(Age - Aj))),
            mortality   = Mu0*exp(Mu1*Age)
        )
      })
}
```

In the Medfly model the specification of the development rate is obviously trivial, as age is the only i-state variable. Furthermore, the code fragment above implements the function $\beta(a) = \beta_0 e^{-\beta_1(a-A_j)}$ for fecundity and the function $\mu(a) = \mu_0 e^{\mu_1 a}$ for the mortality, which is not influenced by the life stage specifically and is only age-dependent.

> Refer to the remarks in the discussion of the function `LifeStageEndings()` concerning the dependence on the individual's state-at-birth.

3.3.1.7 Optional discrete changes at stage boundaries

Even though not listed among the basic assumptions of the PSPM in the beginning of this chapter, it is permissible to have discrete changes or jumps in the individual state variables at the transition between two consecutive life stages. If these occur, they should be specified in the function `DiscreteChanges()`. This function is not relevant in case of the Medfly model, in which case it can simply be left away or commented out.

Code block 3.3.1.7

```
DiscreteChanges <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(E, pars)),{
       # No discrete changes in this problem, function is commented out, which
       # would be equivalent to returning a copy of the input argument 'istate'
       istate
     })
}
```

If defined, the function `DiscreteChanges()` is called whenever a transition between two consecutive life stages is reached during the integration over the individual life history. The function should return a vector of length equal to the number of i-state variables. Each element should specify the value of the particular i-state variable after the transition to the current state. In case the model accounts for multiple, structured populations this function should return a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables.

It should be noted that the value of the variable `lifestage` indicates the life stage that is entered, that is, following the current stage boundary. This routine will hence never be called with a value of one of the elements `lifestage` equal to 1. The discrete changes in the individual state variables have to be implemented by assigning new values to the variables `istate`. These assignments may as before depend on the life stage that is entered, as specified by the variable `lifestage`, the (old values) of the individual state variables, contained in the argument `istate`, and possibly on the individual's state-at-birth, specified in the argument `birthstate`. If no assignment of a value to `istate` is implemented, that particular individual state variable will keep its current value.

3.3.2 Implementation of the model in `C`

The implementation of the Medfly model in `C` requires the specification of 10 pieces of `C`-code that can be subdivided into two different groups:

- Problem dimensions, numerical settings and model parameters
- Definition of the individual life history functions, such as development (growth), fecundity and mortality.

The pieces of `C`-code are discussed in detail in the next 10 subsections. The code can be found in the file `Medfly.h`, which can be opened by executing the command `showpspm("Medfly.h")`. To implement your own model you only need a basic understanding of `C`, which programming language I will not further discuss here. It is advisable to use one of the example models, which can be listed using the utility function `showpspm()`, as a basis for the implementation. To do so, you can copy the contents of the file `Medfly.h` to a new file in `Rstudio`'s built-in editor and save this new file with a new name. The extension of your model-specific file should however remain `'.h'`.

The software allows for the analysis of models with multiple structured populations, each of which consists of individuals that are characterized by a finite number of individual state variables. The number of state variables characterizing an individual should, however, be the same for each of the structured populations in the model. Furthermore, at birth individuals may have one of a finite number of states-at-birth. To distinguish between populations, between individual state variables and between different states-at-birth, in the following the index p will consistently refer to the index of the structured population in the model. Because the dimension setting `POPULATION_NR` is used to specify the number of populations in the model (see the next section), p takes on values in the range $0, 1, \ldots,$ `POPULATION_NR`-1. Similarly, the index i will consistently refer to the index of a particular individual state variable, which should always take values in the range $0, 1, \ldots,$ `I_STATE_DIM`-1, given

that the dimension setting `I_STATE_DIM` determines the number of individual state variables (see the next section).

3.3.2.1 Definition of problem dimensions and optional numerical settings

The code box below defines the different dimensions of the model and the numerical settings to be used in the computations.

Code block 3.3.2.1

```
1 // Dimension settings: Required
  #define POPULATION_NR       1               // Structured consumer population
  #define STAGES              2               // Juvenile & adult
  #define I_STATE_DIM         1               // See below
5 #define PARAMETER_NR        5

  // Numerical settings: Optional (default values adopted otherwise)
  #define MIN_SURVIVAL        1.0E-9          // Survival at which individual is considered dead
  #define MAX_AGE             100000          // Give some absolute maximum for individual age
10
  #define DYTOL               1.0E-7          // Variable tolerance
  #define RHSTOL              1.0E-8          // Function tolerance
```

The software can simultaneously compute the population growth of more than a single population. At the start of the problem file the variable `POPULATION_NR` has to be defined equal to the number of structured populations accounted for in the model. For the Medfly example this is obviously equal to 1 (line 2 in the code box above).

The variable `STAGES` has to be defined equal to the number of life stages that can be distinguished in the individual life history (line 3 in the code box above). While integrating the ODEs for the individual life history numerical problems may occur when the right hand side of the ODEs changes abruptly in value at a certain threshold value of the individual state, as a consequence of discontinuities in the development rate, the mortality rate or the fecundity. Each of such thresholds in the individual life history should be distinguished as a stage boundary. In the Medfly model the fecundity $\beta(a)$ changes from 0 just before $a = A_J$ to β_0 at $a = A_J$ and $\beta_0 \exp(-\beta_1(a - A_j))$ at larger ages. At $a = A_j$ $\beta(a)$ thus exhibits a discontinuity, which separates the juvenile and the adult stage from each other. The variable `STAGES` is therefore set equal to 2.

The variable `I_STATE_DIM` (line 4 in the code box above) defines the dimension of the individual state. As only age characterizes the individuals in the Medfly model, this variable is defined equal to 1.

The last required parameter that has to be specified is the number of parameters in the model (set in line 5 in the code box above). In the Medfly model this equals 5 (β_0, β_1, A_j, μ_0 and μ_1).

The remaining definitions in the code box are all optional and can be left away. A list of all possible variables that can be changed by a definition in this code section is provided in section 9.3. The variable `MIN_SURVIVAL` determines the threshold of the survival probability below which an individual is considered dead. The integration over the individual life history stops whenever the survival probability falls below this threshold value. In the code above (line 8) the minimum survival is set to 10^{-9}, which is in fact the default value and is hence superfluous. Note that the value of `MIN_SURVIVAL` can *not* be set equal to 0. As an alternative to using 0 `MIN_SURVIVAL` can be set to a very small value like 10^{-100}.

The variable `MAX_AGE` (line 9 in the code box above) can be used as an alternative to determine the end of an individual life and to stop the integration over the individual life history. In the Medfly model there is no maximum individual age and hence the variable is set to a very high value (100000), which the individuals will never reach, because before that age their survival probability has already dropped below its threshold value (10^{-9}).

The remaining two quantities `DYTOL` and `RHSTOL` determine whether a solution has been found. In general, both demographic analysis as well as equilibrium analysis of PSPMs boils down to solving

a system of nonlinear equations that can be represented as $G(y) = 0$ for a set of unknowns y in an iterative manner. The subsequent estimates of the solution in the Newton iterations can be labeled as y_p and y_{p+1}. A solution is now considered to be located if both of the following conditions hold:

$$\|y_{p+1} - y_p\| < \epsilon_y$$

$$\|G(y_{p+1})\| < \epsilon_G$$

where $\|.\|$ refers to the Euclidean norm. `DYTOL` and `RHSTOL` are the quantities ϵ_y and ϵ_G, respectively. Increasing (decreasing) their value leads to easier (harder) acceptance of a set of unknowns as a solution to the system of equations $G(y) = 0$. The definition of these two accuracies in the code box above is in fact superfluous as they are defined equal to their default values (see section 9.3).

3.3.2.2 Definition of parameter names and values

The code box below assigns each of the model parameters a meaningful name and a default value.

Code block 3.3.2.2

```
1 // Descriptive names of parameters in parameter array (at least two parameters are required)
  char  *parameternames[PARAMETER_NR] =
      { "Beta0", "Beta1", "AJ", "Mu0", "Mu1"};

5 // Default values of all parameters
  double  parameter[PARAMETER_NR] =
      {47.0, 0.04, 11.0, 0.00095, 0.0581};
```

Model parameter values are stored by the program in the vector variable `parameter`. The lines 2-3 above assign each of the elements this vector a more meaningful, model-specific name. These name strings can not be used in the remaining parts of the model implementation, they only serve to make the output files produced by the program more readable. These output files contain a small header text indicating among other details which parameter values were used for the computation of the results contained in the output file. In this report the parameter names are listed together with their value. To adapt the above code to a different model, the code on line 2 of the code box above should remain the same, only change line 3 as needed (possibly extending it over multiple lines in case there are many parameters).

The default values to use for the model parameters are specified by the declaration of the vector `parameter[PARAMETR_NR]` on line 6-7 of the previous code box. The values should be specified as a comma-separated array of values within braces (don't forget the closing semi-colon at the end of the statement!). To adapt the above code to a different model, the code on line 6 of the code box above should remain the same, only change line 7 as needed (possibly extending it over multiple lines in case there are many parameters).

3.3.2.3 Definition of aliases to simplify implementation

The code box below defines aliases for program variables used in the `C`-implementation of the model, such that they are more easily identified with the model ingredients. Defining these aliases is optional but strongly advised as it makes model implementation more straightforward.

Code block 3.3.2.3

```
1 // Aliases definitions for all istate variables
  #define AGE                 istate[0][0]
```

```
// Aliases definitions for all parameters
#define BETA0               parameter[ 0]   // Default: 47.0
#define BETA1               parameter[ 1]   // Default: 0.04
#define AJ                  parameter[ 2]   // Default: 11.0
#define MU0                 parameter[ 3]   // Default: 0.00095
#define MU1                 parameter[ 4]   // Default: 0.0581
```

The developmental rates in individual state, fecundity and mortality in any model depend on the individual state itself, on the individual's state at birth and on model parameters. The value of the individual state variables at a particular age are always referred to with the program variable `istate[`p`][`i`]`, where the index p refers to the number of the population and the index i refers to the number of the individual state variables. Notice that in `C` array indices run from 0 (as opposed to 1 like in `R`)! Similarly, the value of the individual's state variables at birth are always referred to with the program variable `birthstate[`p`][`i`]`. In case there are multiple populations and/or more than a single individual state variable, it is up to the user to keep track of which index pertains to which population or individual state variable. In the Medfly model there is only a single population and a single individual state variable, while the state at birth is rather irrelevant as it equals age 0. Therefore, `istate[0][0]` is the only program quantity to give a more meaningful name (line 2 in the code box above).

As discussed in the previous section all model parameters are contained in a vector named `parameter` in the code. Which element of this vector represents which model-specific parameter is up to the user. To prevent mixing up the interpretation of the different vector elements and hence to prevent mistakes, it is strongly advised to define meaningful, model-specific aliases for each of the elements of the vector `parameter` as is illustrated in lines 5-9 in the code box above. It is best to avoid completely the direct use of the program variable `parameter` in any part of the model specification and only use the models-specific aliases.

> As can be seen in the code block above all aliases for program variables used in the `C`-implementation of the model are names in capitals. It is advisable to use only capitals when introducing these aliases (or global variables if they are needed) to avoid any conflict between these aliases and variables that defined elsewhere in any of the `C` files with numerical routines that are included in the package.

3.3.2.4 Specifying the number of possible states-at-birth

The first routine to be implemented for a particular life history model defines for every population in the model the number of possible states-at-birth that an individual can be born with (i.e. the value of the size m of the set $\{\phi_1, \ldots, \phi_m\}$).

Code block 3.3.2.4

```
/*
 * Specify the number of states at birth for the individuals in all structured
 * populations in the problem in the vector BirthStates[].
 */

void SetBirthStates(int BirthStates[POPULATION_NR], double E[])
{
  BirthStates[0] = 1;

  return;
}
```

For each population with index p the variable `BirthStates[`p`]` has to be set to the number of possible states at birth. Because individual age is the only *i-state* variable the Medfly model, the state-at-birth is unique and hence `BirthStates[0]` is set to 1.

Note that different populations may have different numbers of states-at-birth. `BirthStates[`p`]` hence does not need to be the same for all p.

3.3.2.5 Specifying the value of all possible states-at-birth

The next routine to implement defines for every possible state-at-birth with index j the actual value of the different individual state variables at birth $\boldsymbol{\phi}_j = \left(\phi_{j1}, \ldots, \phi_{jk}\right)$:

Code block 3.3.2.5

```
/*
 * Specify all the possible states at birth for all individuals in all
 * structured populations in the problem. BirthStateNr represents the index of
 * the state of birth to be specified. Each state at birth should be a single,
 * constant value for each i-state variable.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
{
  AGE    = 0.0;

  return;
}
```

For every population ($p = 0, 1, \ldots,$ `POPULATION_NR`-1) the value of each individual state variable `istate[`p`][`i`]` ($i = 0, 1, \ldots,$ `I_STATE_DIM`-1) has to be assigned a unique value, from which individual development will start at age 0. As shown in the example of the Medfly model, if the life history depends on the age of the individual, age should be explicitly included as one of the individual state variables. The program does not automatically include individual age in its characterization of the individual state, even though integration over the entire life history (as a function of age) is carried out. For the Medfly model age is the only individual state variables and set to 0 at birth.

The routine `StateAtBirth()` will be called as many times as there are possible states-at-birth. The variable `BirthStateNr` indicates the index j of the state-at-birth in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$ for which the values have to be set in the current invocation of the routine. The routine will thus be called with `BirthStateNr` set equal to a value in $0, 1, \ldots, m-1$ (Remember the starting index 0 in `C`!). If there are multiple states-at-birth (`BirthStates[`p`]` $\$ > 1\$$) the definition of the values of the *i-state* variables has to depend explicitly on the index `BirthStateNr` to make the states-at-birth different from each other. Furthermore, if the problem involves multiple structured populations the number of possible states-at-birth can be different for each of them, which might lead to a situation that the routine above is called with a value of the index `BirthStateNr` that is larger than the maximum number of states-at-birth for a particular population (`BirthStateNr` $\geq$ `BirthStates[`p`]`). The program safely ignores such inappropriate state-at-birth specifications.

3.3.2.6 Definition of boundaries between discrete stages

The next routine determines the boundaries between consecutive stages in the individual life history.

Code block 3.3.2.6

```
/*
 * Specify the threshold determining the end point of each discrete life
 * stage in individual life history as function of the i-state variables and
 * the individual's state at birth for all populations in every life stage.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void IntervalLimit(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                   double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                   double limit[POPULATION_NR])
{
  if (lifestage[0] == 0)
    limit[0] = AGE - AJ;

  return;
}
```

In this routine the variable `limit[`p`]` has to be defined, which has as many elements as there are populations ($p = 0 \ldots$ `POPULATION_NR-1`). The life stage that the individual is in at the moment this routine is called, is determined by the variable `lifestage[`p`]`, which has a value of 0 if the individual is in the first life stage and a value of `STAGES`-1 if it is in the last life stage. The element `limit[`p`]` should now indicate when the current life stage as given in `lifestage[`p`]` ends. In particular, the program considers the current life stage to end when `limit[`p`]` turns from negative to positive. For the end of the last life stage the death of old age, either by reaching the maximum age `MAX_AGE` or by reaching the minimum survival threshold `MIN_SURVIVAL`, does not have to be specified separately, the program takes care of that automatically. In the Medfly model therefore only the end of the larval stage has to be specified, as expressed in lines 16-17 of the code box above.

The threshold value that has to be stored and returned to the program in `limit[`p`]` will depend on the individual state variables, possibly on the individual's state-at-birth and will be different for individuals in different life stages. For this reason, the routine `IntervalLimit()` has as arguments `lifestage[]`, specifying the life stage that the individual is currently in, `istate[][]`, the individual state, and `birthstate[][]`, the individual's state-at-birth.

Like the previous routine, this routine will be called as many times as there are possible states-at-birth, because the state-at-birth may influence the threshold between consecutive life stages. The same holds for the routines discussed in sections 3.3.2.7-3.3.2.10 below, which define changes in the *i-state* variables, the fecundity and the mortality of individuals, respectively. In essence, individuals with different states-at-birth are treated as constituting subpopulations within the same structured population. Because of the possible dependence on the state-at-birth the variables `birthstate[][]` and `BirthStateNr` are passed as arguments to this routine and the once discussed in sections 3.3.2.7-3.3.2.10. These arguments contain the values of the *i-state* variables and the index in the set $\{\phi_1, \ldots, \phi_m\}$, respectively, for which the routine is invoked and for which the threshold between consecutive life stages has to be evaluated.

If the problem involves multiple structured populations and the number of possible states-at-birth differs among them, the routine above may be called with a value of the index `BirthStateNr` that is larger than the maximum number of states-at-birth for a particular population (`BirthStateNr` $\geq$ `BirthStates[`p`]`). Although this circumstance may seem confusing, the user does not have to worry about it, as the program is designed to safely ignore such assignments of thresholds between consecutive life stages, changes in the *i-state*

variables, fecundity and mortality of individuals for states-at-birth with index `BirthStateNr` $\geq$ `BirthStates[`p`]` that are inappropriate for the structured population with index p.

Notice that the function header shown in the code box above also contains an array `E[]` as a variable. This array will contain the values of the environment variables during equilibrium computations of PSPMs (see sections 4.1 to 4.4). In demographic analysis of PSPMs this variable is non-functional and is best ignored, using it in a statement inside the routine may even cause the program to crash. The only reason for the presence of this variable in the function header is to keep the function declaration the same for both demographic and equilibrium analysis computations. In principle, the same model-specific file can hence be used for both types of analysis. The variable `E[]` will for the same reasons also be part of the headers of the next 4 routines.

Tip: The more advanced user who wants to perform both demographic and equilibrium analysis using the same model-specific file should notice that the array of environment variables `E[]` can in principle be used inside all the routines, if the dimension `ENVIRON_DIM` determining the number of environment variables has been set (see code box 4.3.2.1 for details). The appropriate value to use for the environment variables should be assigned to the elements `E[`e`]` in the routine `StateAtBirth()` (see code box 3.3.2.5), after which it will keep the same value throughout all the subsequent routines.

3.3.2.7 Specification of continuous individual state development

Code block 3.3.2.7

```
/*
 * Specify the development of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * Notice that the first index of the variables 'istate[][]' and 'development[][]'
 * refers to the number of the structured population, the second index refers
 * to the number of the individual state variable. The interpretation of the
 * latter is up to the user.
 */

void Development(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                 double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                 double development[POPULATION_NR][I_STATE_DIM])
{
  development[0][0] = 1.0;

  return;
}
```

This routine specifies the right-hand side of the ODE:

$$\frac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$$

that determines the continuous development of the individual state variables during the life history. In the Medfly model the specification is obviously trivial. More generally, the value of `development[`p`][`i`]` determines for each structured population p the development in the individual state variable i. Notice that these development rates may differ in different life stages, for example growth in body size may be different for juveniles and adults in case adults invest a lot of energy into reproduction. The development rates should then be specified dependent on the current life stage the individual is in. This current life stage at the moment the routine is evaluated is contained in the variable `lifestage[`p`]`. The development rate may furthermore depend on the individual state variables and possibly on the individual's state-at-birth, which is the reason for `istate[][]`, the individual state, and `birthstate[][]`, the individual's state-at-birth, as arguments to this routine.

Refer to the remarks in section 3.3.2.6 concerning the dependence on the individual's state-at-birth.

3.3.2.8 Specification of discrete individual changes at stage transitions

Even though not listed among the basic assumptions of the PSPM in the beginning of this chapter, it is permissible to have discrete changes or jumps in the individual state variables at the transition between two consecutive life stages. If these occur, they should be programmed in the following routine.

Code block 3.3.2.8

```
/*
 * Specify the possible discrete changes (jumps) in the individual state
 * variables when ENTERING the stage specified by 'lifestage[]'.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void DiscreteChanges(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                     double *birthstate[POPULATION_NR], int BirthStateNr, double E[])
{
  return;
}
```

This routine is not relevant in case of the Medfly model and hence its contents are empty (apart for the necessary `return;` statement).

This routine is called whenever a transition between two consecutive life stages is reached during the integration over the individual life history. It should be noted that the value of the variable `lifestage[`p`]` indicates the life stage that is entered, that is, following the current stage boundary. This routine will hence never be called with a value of one of the elements `lifestage[`p`]` equal to 0. The discrete changes in the individual state variables have to be implemented by assigning new values to the variables `istate[`p`][`i`]`. These assignments may as before depend on the life stage that is entered, as specified by the variable `lifestage[]`, the (old values) of the individual state variables, contained in the argument `istate[][]`, and possibly on the individual's state-at-birth, specified in the argument `birthstate[][]`. If no assignment of a value to `istate[`p`][`i`]` is implemented, that particular individual state variable will keep its current value.

Refer also to the remarks in section 3.3.2.6 concerning the dependence on the individual's state-at-birth.

3.3.2.9 Specification of fecundity

The following routine specifies the fecundity as a function of the individual state. The code fragment below implements the function $\beta(a) = \beta_0 e^{-\beta_1(a-A_j)}$ for the Medfly model. It provides a good example of how to assign a different value for a particular life history rate dependent on the life stage that an individual is in. The same approach can also be used in the other routines specifying the life history rates of individuals.

Code block 3.3.2.9

```
/*
 * Specify the fecundity of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
```

```
 *
 * The number of offspring produced has to be specified for every possible
 * state at birth in the variable 'fecundity[][]'. The first index of this
 * variable refers to the number of the structured population, the second
 * index refers to the number of the birth state.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double *fecundity[POPULATION_NR])
{
 if (lifestage[0] == 1)             // Only for adults
    {
     fecundity[0][0]  = BETA0*exp(-BETA1*(AGE - AJ));
    }
  else
    fecundity[0][0] = 0;

  return;
}
```

In this routine not only the fecundity (i.e. the number of offspring produced per unit time) has to be specified, but also the state-at-birth of the produced offspring. Therefore, this routine has to assign values to the matrix `fecundity[`p`][`j`]`, which determines for the population with index p the number of offspring produced per unit time with state-at-birth with index j in the set $\{\phi_1, \ldots, \phi_m\}$. This fecundity will certainly depend on the life stage that the individual is in (only adults reproduce), which is contained in the argument `lifestage[]`, and on the individual state variables, i.e. the values of the argument `istate[][]`), but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate[][]` and `BirthStateNr`, respectively.

In the most common case of a unique state-at-birth and a single structured population, like in the Medfly model, the only valid indices are $p = 0$ and $j = 0$ and hence only the variable `fecundity[0][0]` has to be assigned.

> For more detailed remarks about models with multiple states-at-birth consult section 3.3.2.6.

3.3.2.10 Specification of mortality

The last routine specifies the mortality as a function of the individual state.

Code block 3.3.2.10

```
/*
 * Specify the mortality of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void Mortality(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double mortality[POPULATION_NR])
```

```
15 {
     mortality[0] = MU0*exp(MU1*AGE);

     return;
   }
```

For each population the corresponding element of the array `mortality[`p`]` should be assigned the mortality rate, possibly dependent on the life stage the individual is in at the moment this routine is called (given in argument `lifestage[]`), the current *i-state* of the individual (given in argument `istate[][]`) and the individual's state-at-birth (current values and index in the set $\{\phi_1, \ldots, \phi_m\}$ given by `birthstate[][]` and `BirthStateNr`, respectively).

In the Medfly model the mortality is not influenced by the life stage specifically and is only age-dependent. The line 16 in the code box above implements the function $\mu(a) = \mu_0 e^{\mu_1 a}$.

> Refer to the remarks in section 3.3.2.6 concerning the dependence on the individual's state-at-birth.

3.4 Model analysis

3.4.1 Executing the `PSPMdemo` function

Once the model has been implemented, you can proceed carrying out its analysis, which in the simplest approach is performed by calling the function `PSPMdemo` with the name of the file specifying the PSPM passed as a string argument. It is unnecessary to include the extension `'.R'` or `'.h'` as part of the file name, the `PSPMdemo` function will automatically try to locate the appropriate file, checking first for a file implemented in `C` (with an extension `'.h'`) and subsequently for a file implemented in `R` (with an extension `'.R'`). If both a file with an extension `'.h'` and a file with an extension `'.R'` are found, the program will use the first one. The program can be forced to use the file with an extension `'.R'` by including the extension explicitly as part of the file name. Therefore, the invocation `PSPMdemo("Medfly")` is identical to `PSPMdemo("Medfly.h")` if the model is implemented in `C` and to `PSPMdemo("Medfly.R")` if the model is implemented in `R`. Furthermore, if the file specifying the PSPM can not be found in the current directory, the `PSPMdemo` function will ask the user to search in the package directory for a model file with the specified name.

These two calls of the `PSPMdemo`-function will give the same output as the following invocation of the function with the two optional arguments `clean=TRUE` and `force=TRUE`:

Command box 3.4.1.A

```
1  > PSPMdemo("Medfly", clean=TRUE, force=TRUE)

   Building executable  /Users/andre/programs/PSPM analysis/Tests/Medflydemo.so ...

5  <...compilation output suppressed in this box...>

   #
   # Executing : PSPMdemo("Medfly", NULL, NULL, NULL)
   #
10 # Parameter values  :
   #
   #   Beta0     :  47             Beta1     :  0.04           AJ        :  11
   #   Mu0       :  0.00095        Mu1       :  0.0581
   #
15 #   1:PGR[ 0]     2:Tc[ 0]  3:S[ 0][ 0]  4:S[ 0][ 1]  5:S[ 0][ 2]  6:S[ 0][ 3]  7:S[ 0][ 4]
   #
      0.41905662  13.16725955   0.00161586  -0.16459366  -0.03198198  -1.52635957  -0.01132532
```

The `PSPMdemo` function first compiles the model-specific file using the `R` command `R CMD SHLIB` into a dynamically loadable library file, which can subsequently be executed. The output of this compilation step is system specific and hence suppressed in the command box above. The compilation step is only carried out when the executable (on Mac OS X and Linux systems called `Medflydemo.so`) does not exist, or when the model-specific has been changed since the last compilation of the executable. Furthermore, the compilation step is forced by the invocation of `PSPMdemo` with the additional argument `force=TRUE` as in the command box shown above.

When the `PSPMdemo` function is invoked in the way shown above the output of the computation is only printed to the console, the function does not return any variables or results (as is clear from the boxed material above). Apart from printing the exact command-line that has been used to start the computation, the values of the parameters are printed using the meaningful, model-specific names that are used as labels of the vector elements of the variable `DefaultParameters` when the model is implemented in `R` (see section 3.3.1.3) or when the model is implemented in `C` that are defined as the string elements in the variable `parameternames` (see section 3.3.2.2). Notice that 3 additional but optional arguments to the function `PSPMdemo` are reported as being set to `NULL`, meaning they were not defined.

The numerical output generated by the model is printed as a single line of numbers. The first column of this output contains the computed population growth rate. The second column contains the generation time in the stable population state, which corresponds to the average age at reproduction in the exponentially growing population and is defined as:

$$\int_0^\infty a\, e^{-ra} \beta(\boldsymbol{\chi}(a))\, \mathcal{F}(a)\, da$$

in which r represents the population growth rate, $\beta(\boldsymbol{\chi}(a))$ the fecundity of an individual with individual state $\boldsymbol{\chi}(a)$ at age a and $\mathcal{F}(a)$ the probability that an individual survived up to age a. The following columns show the sensitivity of the population growth rate with respect to the model parameters in the order as they are defined in the variable `DefaultParameters` when the model is implemented in `R` (see section 3.3.1.3) or in the variable `parameter` (see section 3.3.2.2) when the model is defined in `C`. For the Medfly model these are the sensitivities to the 5 model parameters that are printed directly above.

The second method to invoke the `PSPMdemo` function is with an additional arguments to calculate the population growth rate as a function of one of the model parameters for a range of values of this parameter. This can be achieved by passing as an additional argument to the function a vector of 5 elements of the following form:

`c(`<*index*>,<*starting value*>,<*step size*>,<*minimum value*>,<*maximum value*>`)`

The first element indicates the index of the parameter in the vector `DefaultParameters` (see section 3.3.1.3) or in the array `parameter` (see section 3.3.2.2) to vary, the second element of the array indicates its starting value from which to compute the curve of the population growth rate as a function of the parameter, the third value indicates the step size in the parameter along this curve (which can be either positive or negative), while the final two elements of the array indicate the minimum and maximum value of the parameter. The computation of the curve of the population growth rate as a function of the model parameter stops, whenever the minimum or maximum parameter value is reached.

> When the model is implemented in `R` the name of the vector element in the vector `DefaultParameters` can be used (passed as a string) to identify the parameter to vary, instead of its index.

The following `R` code illustrates this use of the `PSPMdemo` function for the Medfly model by computing the population growth rate as a function of A_j (`parameter[2]` in `C`, `DefaultParameters[3]` in `R`),

starting at the initial and default value of $A_j = 11$ and computing the growth rate for increasing values of the parameter with step size 0.1, while limiting the computation to the interval $11 \leq A_j \leq 20$. Notice that in `C` array indices start at the value 0, whereas in `R` vector indices start at 1. The code below provides the command-line for the Medfly model implemented in `C`.

Command box 3.4.1.B

```
> output <- PSPMdemo("Medfly.h", c(2, 11, 0.1, 11, 20), c(47, 0.04, 11, 0.00095, 0.0581), c("isort", "0"),
+ clean=TRUE, force=TRUE, debug=FALSE)

Building executable  /Users/andre/programs/PSPM analysis/Tests/Medflydemo.so ...

<...compilation output suppressed in this box...>

  1.10000000E+01  4.19056620E-01
  1.11000000E+01  4.15884247E-01
  1.12000000E+01  4.12762685E-01
<...output lines suppressed in this box...>
  1.98000000E+01  2.53879994E-01
  1.99000000E+01  2.52772251E-01
  2.00000000E+01  2.51674454E-01

#
# Executing : PSPMdemo("Medfly", c(2, 11, 0.1, 11, 20), c(47, 0.04, 11, 0.00095, 0.0581), c("isort", "0"))
#
# Parameter values  :
#
#   Beta0     :  47             Beta1     :  0.04           AJ        :  11
#   Mu0       :  0.00095        Mu1       :  0.0581
#
# Index of bifurcation parameter #1                          : 2
#
#        1:AJ    2:PGR[ 0]     3:Tc[ 0]  4:S[ 0][ 0]  5:S[ 0][ 1]  6:S[ 0][ 2]  7:S[ 0][ 3]  8:S[ 0][ 4]
#
> output
$curvedesc
 [1] "#"
 [2] "# Executing : PSPMdemo("Medfly", c(2, 11, 0.1, 11, 20), c(47, 0.04, 11, 0.00095, 0.0581), c("isort", "0"))"
 [3] "#"
 [4] "# Parameter values  : "
 [5] "#"
 [6] "# \tBeta0     :  47           \tBeta1     :  0.04         \tAJ        :  11           "
 [7] "# \tMu0       :  0.00095      \tMu1       :  0.0581       "
 [8] "#"
 [9] "# Index of bifurcation parameter #1                          : 2"
[10] "#"
[11] "#        1:AJ    2:PGR[ 0]     3:Tc[ 0]  4:S[ 0][ 0]  5:S[ 0][ 1]  6:S[ 0][ 2]  7:S[ 0][ 3]  8:S[ 0][ 4]"
[12] "#"

$curvepoints
        V1        V2       V3           V4         V5          V6        V7          V8
 [1,] 11.0 0.4190566 13.16726 0.0016158600 -0.1645937 -0.03198198 -1.526360 -0.01132532
 [2,] 11.1 0.4158843 13.28218 0.0016018800 -0.1642926 -0.03146749 -1.532324 -0.01148047
 [3,] 11.2 0.4127627 13.39705 0.0015881500 -0.1639943 -0.03096572 -1.538315 -0.01163686
<...output lines suppressed in this box...>
[89,] 19.8 0.2538800 23.15013 0.0009190646 -0.1447124 -0.11127611 -2.172537 -0.03082490
[90,] 19.9 0.2527722 23.26222 0.0009146362 -0.1445347 -0.11027447 -2.181510 -0.03112948
[91,] 20.0 0.2516744 23.37427 0.0009102513 -0.1443576 -0.10928690 -2.190525 -0.03143628
```

Some of the intermediate lines of output generated by `R` in this case are suppressed for brevity. When the `PSPMdemo` function is invoked in this way to compute parameter dependence, it generates a single list as output (assigned to the variable `output` in the command box above), which contains two elements, called `curvedesc` and `curvepoints`. The variable `output$curvedesc` contains the description of the executed calculation, which is the textual information that is also printed to the `R`

console at the end of calculations. In fact, the PSPMdemo function prints its report on the calculations by execution of the statement `cat(output$curvedesc, sep=' ')`.

The output variable output$curvepoints is a two-dimensional array containing columns of computed output with as a first column the value of the parameter, the second column the value of the population growth rate for that particular parameter value and the third column the generation time (the average age at reproduction) in the stable population during the exponential growth phase. The subsequent columns represent the sensitivities of the population growth rate to all model parameters, as discussed before. The output of output$curvepoints in the box above shows that the data indeed start at $A_j = 11$ and that the computation is terminated when a value of A_j is reached that exceeds the maximum parameter value specified. The data contained in the output variable can subsequently be used for plotting or for further calculations.

3.4.2 Output files generated by the PSPMdemo function

The PSPMdemo function and module generates 2 output files when the function is only performing a single population growth rate calculation and 3 output files when the population growth rate is computed as a function of a model parameter. The name of these files is always of the form <Modelname>-PGR-<NNNN>.<ext>, in which <Modelname> is the same as the name of the file specifying the model excluding its '.h' or '.R' extension, <NNNN> is a 4-digit number that is unique for the current computation and .<ext> is the extension, which can be either .err, .csb or .out. Hence, the invocation of the PSPMdemo function for the Medfly model, as shown in command box 3.4.1.A, generates the output files Medfly-PGR-0000.err and Medfly-PGR-0000.csb, while the invocation of the PSPMdemo function for the Medfly model, as shown in command box 3.4.1.B, generates three output files: Medfly-PGR-0001.err, Medfly-PGR-0001.csb and Medfly-PGR-0001.out. For the 4-digit number <NNNN> in the file name, the program always finds the lowest positive value that is not in use yet. However, whenever the PSPMdemo function is invoked with the (optional) argument clean=TRUE, the PSPMdemo function deletes all output files that have been generated for the particular model studied (all files called <Modelname>-PGR-<NNNN>.err, <Modelname>-PGR-<NNNN>.csb and <Modelname>-PGR-<NNNN>.out, and hence the 4-digit file identification number will restart at 0000 again. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function PSPMclean(), taking no arguments, to delete all .bif, .err, .csb and .out files and/or all executable files that are present in the current working directory.

The file called <Modelname>-PGR-<NNNN>.err contains information about the numerical progress of the computation. It reports details on the steps take during the Newton iteration, the convergence to the solution, as well as information about the steps taken along the curve that is being computed. This file can be informative in case the computation of a particular curve stops for unknown reasons, but is otherwise of little use.

The file called <Modelname>-PGR-<NNNN>.out contains the same information as is contained in the output variables output$curvedesc and output$curvepoints returned by the PSPMdemo function. The first lines of this file all start with a '#' sign and contain the information about the run performed, which is also contained in output$curvedesc and can be listed by the statement `cat(output$curvedesc, sep='\n')` (see the command boxes in the previous section). Following this descriptive header the file contains columns with computational results that are also contained in the variable output$curvepoints, that is, the parameter values, population growth rates, generation times and sensitivities of the population growth rate to all model parameters. Command box 3.4.1.B provides an example of the type of output generated by the computational module.

The last output file generated during the population growth rate has a name of the form <Modelname>-PGR-<NNNN>.csb and contains information on the stable population distribution for every parameter value for which the population growth rate is computed. This is a binary file, the

content of which can be accessed from R using the function csbread. For example, the file contents of the file Medfly-PGR-0000.csb generated by the computation in command box 3.4.1.B can be listed by:

Command box 3.4.2.A

```
> csbread("Medfly-PGR-0000.csb")

States in file Medfly-PGR-0000.csb:

    1: State-1.100000E+01
    2: State-1.110000E+01
    3: State-1.120000E+01
<...output lines suppressed in this box...>
   89: State-1.980000E+01
   90: State-1.990000E+01
   91: State-2.000000E+01
```

Invoking the function csbread with only the file name as argument provides a listing of all the population state stored in the file, one for each of the parameter value for which the population growth rate has been computed. The contents each of these population states can be listed by providing as a second argument to the function csbread either the index of the particular population state in the file or a string with its descriptive name. Therefore, the commands `csbread("Medfly-PGR-0000.csb", 3)` and `csbread("Medfly-PGR-0000.csb", "State-1.120000E+01")` are equivalent, producing the following output:

Command box 3.4.2.B

```
> csbread("Medfly-PGR-0000.csb", 3)
$BifPars
[1] 11.2

$Parameters
[1] 47.00000  0.04000 11.20000  0.00095  0.05810

$PGR
[1] 0.4127627

$Pop00_BirthStates
     Istate00
[1,]        0

$Pop00
         StableDist   Istate00   ReproVal
  [1,] 1.000000e+00  0.0000000   1.000000
  [2,] 8.129859e-01  0.5004307   1.230034
  [3,] 6.609367e-01  1.0008614   1.513004
<...output lines suppressed in this box...>
 [98,] 1.535699e-09 48.5417781   8.421419
 [99,] 1.239033e-09 49.0422089   4.603061
[100,] 1.000000e-09 49.5413326   0.000000
```

This population state, with index 3 and descriptive name State_1_120000E01, pertains to the parameter value $A_j = 11.2$ as its name suggests. The state is returned by the function csbread as a list, which can hence be assigned to a variable in R with the command state<-csbread("Medfly-PGR-0000.csb", 3). The first element of this list (called $BifPars) contains the value of the bifurcation parameter for this particular state. The second element, an array called $Parameters, contains the values of all the model parameters for which the population growth rate has been computed, while the third member of the list contains the computed population growth rate. In the case of the Medfly model this is a single scalar value, but if the population growth rate is computed for more than one population at a time, the population growth rate values are making up an array as well. The two subsequent elements characterize the stable population distribution, of

which the first (called `$Pop00_BirthStates`) specifies the state at birth of the individuals. The other (called `$Pop00`) is a two-dimensional array containing in the first column the density profile of the stable population, in the second column the individual state variable and the reproductive value of the individuals in the last column, as shown in the box above.

In the Medfly model individuals are only characterized by their age and hence their is only a single column with individual state variables. If individuals are characterized by more than a single individual state variable the values of these follow in additional columns of the two-dimensional array `$Pop00`. The last column of this array always contains the reproductive value of an individual. For an explanation of the reproductive value and its computation I refer to De Roos (2008).

3.4.3 Required and optional arguments of `PSPMdemo`

As shown in `R` command boxes in section 3.4.1 at least one argument has to be passed to the `PSPMdemo` function, the base name of the file with the model specification, that is without its `'.h'` or `'.R'` extension. It is the only obligatory argument, all other arguments that can be passed to the `PSPMdemo` function are optional. If the model name is the only argument, the function computes the population growth rate for the default parameter set defined in the `'.h'` or `'.R'` file (see command box 3.4.1.A).

The optional second argument to the `PSPMdemo` function is used to compute the population growth rate over a range of a particular model parameter, as shown in and discussed following command box 3.4.1.B.

The optional third argument of the `PSPMdemo` function is a 1-dimensional array of model parameter values. When used, this array should have the same length as the number of parameters in the model (the length of the variable `DefaultParameters` when the model is implemented in `R` or the value of the constant `PARAMETER_NR` when implemented in `C`). When of this length the values will replace the default values of the parameters that are listed in the model specification file. If the array used for this third argument is not of the correct length, it will simply be ignored.

The optional fourth argument of the `PSPMdemo` function is a vector containing possible options that modify the behavior of the computational module. A useful option is the `"test"` option, which can be passed to the computational module by using the argument `c("test")` as fourth argument to the `PSPMdemo` function. This invokes the computational module in testing mode, which implies that only a single integration of the individual life history is carried out and no iteration to locate the population growth rate is performed. In testing mode the computational module reports on the dynamics of the individual state variables, the survival and the expected number of offspring produced by an individual during its different life stage as well as over its entire life. Testing mode is very useful to discover whether or not the model implementation gives sensible results or not.

The other possible element of the option vector that modifies the behavior of the computational module is the `"isort"` option, which can be passed to the computational module by using for example `c("isort", "1")` as fourth argument to the `PSPMdemo` function (as shown in command box 3.4.1.B). This option modifies the population state output that is stored in the output file, which when using the package in `R` is a binary file with a name of the form `<Modelname>-PGR-<NNNN>.csb` (see above). By default the computational module reports the information about the stable population state distribution and the reproductive value for 100 equidistant values of the first individual state variable. More specifically, the range of the first individual state variable that is covered during the entire life of an individual until the moment that it is considered dead (i.e. the maximum age or the minimum survival threshold has been reached) is subdivided into 100 equidistant intervals and the population density function, individual state variables and reproductive value are computed at each of these 100 nodal values of the first state variable. By using the option `"isort"` the default choice to use the first individual state variable for this subdivision can be changed to the second, third, and so on. Notice though, that the obligatory index value that has to be passed together with the use of the `"isort"` option follows the `C`-convention of ordering arrays starting at 0 (as opposed to `R` where array indices

start at 1). Therefore, passing `c("isort", "0")` as option array to the `PSPMdemo` function is the same as the default behavior: the first individual state variable is used for the subdivision and ordering of the population state distribution, while passing `c("isort", "1")` would use the second individual state variable for this purpose. Also notice that the default number of subdivisions of the individual state variable and hence the number of nodal values for which the population state distribution is reported can be changed by including a statement of the form

```
#define COHORT_NR               200
```

among the definitions of the numerical settings in the model specification when the model is implemented in `C` (see section 3.3.2.1 and section 9.3) or as one of the elements of the vector `NumericalOptions`:

```
NumericalOptions <- c(...
                      COHORT_NR     = 200,
                      ...)
```

if the model is implemented in `R` (see section 3.3.1.2 and section 9.3).

If necessary the options `"test"` and `"isort"` can be combined, for example, as

```
c("isort", "1","test")
```

Or equivalently,

```
c("isort", "1","test")
```

Three other optional arguments can be passed to the `PSPMdemo` function: `clean`, `force` and `debug`. These are all boolean arguments that hence have to be passed to the `PSPMdemo` function as *<option name>*`=TRUE` or *<option name>*`=FALSE`, the latter being the default value of all options (Specifying these options as argument is hence only useful when setting them equal to `TRUE`). Unlike the previous arguments, which all modify the computations to be performed, these options modify the behavior of the `PSPMdemo` function itself, in particular the compilation of the model specific file into a dynamic library module that can be executed from `R`. Also unlike all the previous arguments that can be passed, these arguments can be passed in any order and at any position, the `PSPMdemo` function will filter these 3 optional arguments from the argument list before passing the filtered argument list to the computational routine.

- Option `clean`: When `clean=TRUE` is passed as argument, this argument instructs the `PSPMdemo` function to delete all result files that have been generated during previous calculations with the model. These result files have names of the form `<Modelname>-<Type>-<NNNN>.err`, `<Modelname>-<Type>-<NNNN>.csb` and `<Modelname>-<Type>-<NNNN>.out`, in which `<Modelname>` refers to the name of the model (i.e. `Medfly` in the example model presented in previous sections), `<Type>` refers to the type of computation that has been performed, which in the case of `PSPMdemo` equals `PGR`, and `<NNNN>` is a unique number that distinguishes consecutive computations of the same type of curve with the same model. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function `PSPMclean()`, to delete all `.bif`, `.err`, `.csb` and `.out` files and/or all executable files that are present in the current working directory.
- Option `force`: When `force=TRUE` is passed as argument, it instructs the `PSPMdemo` function to force re-compilation of the model specific file into a dynamic library module that can be executed by `R`. This option will usually not be needed by normal users, as the `PSPMdemo` function automatically recompiles the computational module when the model specific file with an `'.h'` or `'.R'` extension is more recently changed than the compiled dynamic library file. However, if for some unclear reason this automatic recompilation fails, the `force` option can be used to initiate re-compilation.

- Option `debug`: When `debug=TRUE` is passed as argument, it instructs the `PSPMdemo` function to turn on debugging flags while compiling the model specific file into a dynamic library module. This option can be useful to detect programming mistakes in the model-specific file that are otherwise hard to track down. The downside is that depending on the version of `R` that is used, turning on debugging flags during compilation may generate a lot of output, including warnings about standard files of the operating system that are perfectly correct. It is hence not so easy to spot among all these messages the warnings that relate to the model-specific code that has been implemented.

Chapter 4

Equilibrium analysis of nonlinear PSPM

4.1 Model formulation and ingredients

The core of a nonlinear PSPM consists of a model of the individual life history that is based on the following assumptions:

- Individuals are characterized by their *individual* or *i-state*, which is a (finite) set of physiological characteristics (traits such as age, size, sex, energy reserves):

  $$\boldsymbol{\chi} = (\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_k) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

- Individuals are born with an *i-state* $\boldsymbol{\chi}_b$ that is one of a finite set of possible states at birth:

  $$\boldsymbol{\chi}_b \in \{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$$

  with each potential state at birth $\boldsymbol{\phi}_j$ a valid *i-state*:

  $$\boldsymbol{\phi}_j = (\phi_{j1}, \ldots, \phi_{jk}) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

- Individuals are assumed to live in an environment characterized by a (finite) set of *environment variables*:

  $$E = (E_1, \ldots, E_n) \in \mathbb{R}^{\mathrm{n}}$$

  Environment variables can include independent quantities like resource density and density of predators, but also density-dependent measures like total number of individuals or biomass in the population

- Individual and environmental state variables determine, possibly together with the individual's state-at-birth, the individual life history (development, reproduction, mortality)

- Development follows a deterministic process that is continuous in time:

  $$\frac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$$

- Reproduction is a function $\beta(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ of the individual state, the individual's state-at-birth and its environment

- Mortality is a function $\mu(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ of the individual state, the individual's state-at-birth and its environment
- Individuals have an impact $\gamma(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ on their environment
- Environment variables may follow autonomous dynamics in absence of individuals:

  $$\frac{dE_i}{dt} \;=\; G(E)$$

  or be a density-dependent function of the population:

  $$E_i(t) \;=\; \int_\Omega \gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, n(t, \boldsymbol{\chi})\, d\boldsymbol{\chi}$$

Most of the above assumptions are characteristic for the entire class of non-linear PSPMs. The most restrictive of these assumptions concerns the deterministic development process. Biologically, this assumption implies that all individuals that are born with the same state at birth will remain identical throughout their life and will hence not diverge in their *i-state* characteristics. Reproduction and mortality on the other hand are at an individual level considered as stochastic processes, which translate to per-capita rate functions at the population level, given that it is assumed that the number of individuals is large (technically speaking the number of individuals for every possible *i-state*).

4.2 An example model for equilibrium and bifurcation analysis

The steps needed for the implementation of a particular nonlinear PSPM will be discussed using a simple, tritrophic model for the basic resource, a size-structured consumer population and an unstructured predator population, which is discussed in De Roos and Persson (2002). The individual life history of the consumer in this model is dependent on the individual body length ℓ, the resource density R and the predator density P. The PSPM for this can be described by the following set of ordinary and partial differential equations for the resource density R, the consumer size (i.e. length) distribution $c(t, \ell)$ and the predator density P:

$$\frac{dR}{dt} \;=\; \rho\,(R_{max} \;-\; R) \;-\; \int_{\ell_b}^{\ell_m} I(\ell, R)\, c(t, \ell)\, d\ell$$

$$\frac{\partial c(t,\ell)}{\partial t} \;+\; \frac{\partial g(\ell, R)\, c(t,\ell)}{\partial \ell} \;=\; -\mu(\ell, P)\, c(t, \ell)$$

$$g(\ell, R)\, c(t, \ell_b) \;=\; \int_{\ell_j}^{\ell_m} \beta(\ell, R)\, c(t, \ell)\, d\ell$$

$$\frac{dP}{dt} \;=\; \left(\epsilon \frac{aB}{1 + T_h B} \;-\; \delta\right) P$$

$$B \;=\; \int_{\ell_b}^{\ell_v} \omega \ell^3\, c(t, \ell)\, d\ell$$

In this model the resource follows semi-chemostat dynamics in the absence of consumers. Consumers forage on the resource following the length-dependent function $I(\ell, R)$, defined as:

$$I(\ell, R) = I_m \ell^2 \frac{R}{R_h + R}$$

Consumers grow in length from their size at birth ℓ_b to their absolute maximum size ℓ_m with a growth rate $g(\ell, R)$ and produce offspring at a rate $\beta(\ell, R)$, which rates both depend on the consumer length itself and the current resource density:

$$g(\ell, R) = \gamma \left(\ell_m \frac{R}{R_h + R} - \ell \right)$$

$$\beta(\ell, R) = \begin{cases} 0 & \text{if } \ell < \ell_j \\ r_m \ell^2 \dfrac{R}{R_h + R} & \text{otherwise} \end{cases}$$

Consumers experience a mortality rate $\mu(\ell, P)$ dependent on their own length and the current predator density:

$$\mu(\ell, P) = \begin{cases} \mu_b + \dfrac{aP}{1 + T_h B} & \text{if } \ell < \ell_v \\ \mu_b & \text{otherwise} \end{cases}$$

From these equations it can be inferred that predators forage only on consumers with a length between the length at birth ℓ_b and ℓ_v. Larger consumers are invulnerable to predation. The quantity B represents the biomass of consumers in this vulnerable size range, which biomass governs the growth rate of the predator population following a type II functional response.

As listed in section 4.1, individuals are assumed to be born with an *i-state* $\boldsymbol{\chi}_b$ that is one of a finite set of possible states-at-birth, each of which is a valid *i-state*:

$$\boldsymbol{\chi}_b \in \{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}, \qquad \boldsymbol{\phi}_j = (\phi_{j1}, \ldots, \phi_{jk}) \in \Omega \subset \mathbb{R}^{\mathrm{k}}$$

Given that in the PNAS model all individuals are born with age 0 and length $\ell = \ell_b$, all individuals have the same state at birth and hence $m = 1$. The option to specify multiple states-at-birth is hence not relevant for the example model discussed in this implementation chapter. This might hold more generally; most if not all physiologically structured population models that have been reported on in the literature so far are characterised by such a unique state-at-birth for all individuals. Nonetheless, the option to define multiple states-at-birth opens up some interesting research possibilities, which are discussed further in section 9.1.

Since models involving multiple states-at-birth are not very common, information that relates to this option will be distinguished in the text by setting them apart in paragraphs like this one. The index j will be used to refer to the index of a particular state-at-birth in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$. The number m of possible states-at-birth is set dynamically in the model file.

4.3 Implementation of the example model

4.3.1 Implementation of the model in R

The implementation of this model, which I will refer to as the PNAS model, requires the specification of 4 constants and 4 functions describing the life history. The necessary pieces of R-code are discussed in detail in the next subsections. The code can be found in the file `PNAS2002.R`, which can be opened by executing the command `showpspm("PNAS2002.R")` at the command line. To implement your own model it

is advisable to use one of the example models, which can be listed using the utility function `showpspm()`, as a basis for the implementation. To do so, you can copy the contents of the file `PNAS2002.R` to a new file in `Rstudio`'s built-in editor and save this new file with a new name. The extension of your model-specific file should however remain `'.R'`.

4.3.1.1 Problem dimensions

The first variable to define, `PSPMdimensions`, is a vector with the named elements `PopulationNr`, `IStateDimension`, `LifeHistoryStages` and `ImpactDimension` that specify the dimensions of the model:

Code block 4.3.1.1

```
1 PSPMdimensions <- c(PopulationNr = 1, IStateDimension = 2, LifeHistoryStages = 3, ImpactDimension = 4)
```

The software allows for the analysis of models with multiple structured populations, each of which consists of individuals that are characterized by a finite number of individual state variables. The number of state variables characterizing an individual should, however, be the same for each of the structured populations in the model. The vector element `PopulationNr` of `PSPMdimensions` has to be defined equal to the number of structured populations accounted for in the model. For the PNAS2002 example this is obviously equal to 1. The vector element `IStateDimension` of `PSPMdimensions` defines the dimension of the individual state. In the PNAS2002 model this variable is defined equal to 2 as the individual age is included among the individual state variable in addition to the individual length.

The element `LifeHistoryStages` of `PSPMdimensions` has to be defined equal to the number of life stages that can be distinguished in the individual life history. While integrating the ODEs for the individual life history numerical problems may occur when the right hand side of the ODEs changes abruptly in value at a certain threshold value of the individual state, as a consequence of discontinuities in the development rate, the mortality rate or the fecundity. Each of such thresholds in the life history should be distinguished as a stage boundary. In the PNAS model the mortality changes discontinuously at $\ell = \ell_v$, while the fecundity changes discontinuously at $\ell = \ell_j$. Three life stages can hence be distinguished: vulnerable juveniles, invulnerable juveniles and adults, and the changes in the life history rates are indeed abrupt at the transition boundaries between these stages. The element `LifeHistoryStages` of `PSPMdimensions` is therefore set equal to 3.

Finally, the `ImpactDimension` of `PSPMdimensions` has to be defined equal to the number of functions that represent the impact of an individual on its environment. In the PNAS model this feedback of an individual consumer on its environment consists of its grazing rate $I(\ell, R)$ and the biomass-length relation $\omega\ell^3$ determining the biomass of vulnerable consumers, as it represents food for predators. Therefore, the value of `ImpactDimension` should be at least set equal to 2. However, all interaction functions are also saved to the output file generated during a computation. The interaction functions can hence be conveniently used to produce arbitrary output quantities of the form

$$\int_\Omega h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, \tilde{n}(\boldsymbol{\chi})\, d\boldsymbol{\chi}$$

where $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is an interaction (weighing) function that can depend on the values of the individual state $\boldsymbol{\chi}$, the state-at-birth of individuals $\boldsymbol{\chi}_b$ and the values of the environment variables E, and $\tilde{n}(\boldsymbol{\chi})$ is the stable population distribution in equilibrium. Such quantities can therefore represent the total population density in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$), the total population biomass (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ equals the biomass of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$) or the total population birth rate in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is the fecundity of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$). In the PNAS model I want in addition to the biomass of

vulnerable consumers, also the biomass of non-vulnerable juvenile consumers and the biomass of adult consumers as output of the model and hence have set the value of the element `ImpactDimension` in `PSPMdimensions` equal to 4.

4.3.1.2 Optional numerical settings

The next variable specified in the `PNAS2002.R` file is the vector `NumericalOptions`, which can contain a variable number of named vector elements:

Code block 4.3.1.2

```
NumericalOptions <- c(MIN_SURVIVAL  = 1.0E-9,   # Survival at which individual is considered dead
                      MAX_AGE       = 100000,   # Give some absolute maximum for individual age
                      DYTOL         = 1.0E-7,   # Variable tolerance
                      RHSTOL        = 1.0E-6,   # Function tolerance
                      ALLOWNEGATIVE = 0,        # Negative solution values allowed?
                      COHORT_NR     = 100)      # Number of cohorts in population state output
```

The specification of `NumericalOptions` is optional and can be left out, if default values are acceptable. A list of all possible vector elements that can be included in the `NumericalOptions` variable is provided in section 9.3.

The vector element `MIN_SURVIVAL` of `NumericalOptions` determines the threshold of the survival probability below which an individual is considered dead. The integration over the individual life history stops whenever the survival probability falls below this threshold value. In the code above the minimum survival is set to 10^{-9}, which is in fact the default value and is hence superfluous. Note that the value of `MIN_SURVIVAL` can *not* be set equal to 0. As an alternative to using 0 `MIN_SURVIVAL` can be set to a very small value like 10^{-100}.

The vector element `MAX_AGE` of `NumericalOptions` can be used as an alternative to determine the end of an individual life and to stop the integration over the individual life history. In the PNAS2002 model there is no maximum individual age and hence the variable is set to a very high value (100000), which the individuals will never reach, because before that age their survival probability has already dropped below its threshold value (10^{-9}).

The two vector elements `DYTOL` and `RHSTOL` of `NumericalOptions` determine whether a solution has been found. In general, both demographic analysis as well as equilibrium analysis of PSPMs boils down to solving a system of nonlinear equations that can be represented as $G(y) = 0$ for a set of unknowns y in iterative manner. The subsequent estimates of the solution in the Newton iterations can be labeled as y_p and y_{p+1}. A solution is now considered to be located if both of the following conditions hold:

$$\|y_{p+1} - y_p\| < \epsilon_y$$

$$\|G(y_{p+1})\| < \epsilon_G$$

where $\|.\|$ refers to the Euclidean norm. `DYTOL` and `RHSTOL` are the quantities ϵ_y and ϵ_G, respectively. Increasing (decreasing) their value leads to easier (harder) acceptance of a set of unknowns as a solution to the system of equations $G(y) = 0$. The definition of these two accuracies in the code box is in fact superfluous as they are defined equal to their default values (see section 9.3).

The vector element `ALLOWNEGATIVE` of `NumericalOptions` is a flag that can only have a value of 0 or 1 and determines whether or not computations should stop when one of the variables to solve for reaches a negative value. In most population models negative solution values are biologically not relevant and `ALLOWNEGATIVE` is hence set to 0 by default. Line 16 in the code box above is only included to illustrate the use of `ALLOWNEGATIVE` and does not change the value of this variable from its default

value. Most likely, setting `ALLOWNEGATIVE` equal to 1 as opposed to 0 will only be useful in specific cases.

The last vector element `COHORT_NR` of `NumericalOptions` defines the number of cohorts making up the equilibrium population output. During computations of the equilibrium a number of output files will be generated (see section 4.4.5), one of which is a file containing the population equilibrium state for each parameter that the equilibrium values are computed for. The vector element `COHORT_NR` specifies how many cohorts should be used to represent these equilibrium population states. Larger values will generate more detailed representations of the equilibrium population state at the expense of larger file sizes.

4.3.1.3 Names and types of environmental state variables

The next variable defined in the `PNAS2002.R` file is a vector `EnvironmentState` with named elements that specify the environmental state variables in the model. The names of the vector elements can be used in the programming of the life history functions of the model, as shown in the following sections. The value of each of the vector elements should be one of the strings `"PERCAPITARATE"`, `"GENERALODE"` or `"POPULATIONINTEGRAL"` and indicate the nature of the particular environmental state variable. As discussed in section 4.1 environment variables can include independent quantities like resource density and density of predators, but also density-dependent measures like total number of individuals or biomass in the population. The string values of the elements in the vector `EnvironmentState` inform the program about the exact nature of the particular environmental state variable.

The first type of environment variable, indicated with the string `"PERCAPITARATE"`, is one that follows dynamics described by an ordinary differential equation (ODE) and in addition can potentially be 0 in equilibrium. The ODE describing the dynamics of such an environment variable $E_i(t)$ is then of the general form:

$$\frac{dE_i}{dt} = G(E, I)\, E_i$$

in which E is the vector of environment variables, I is the vector of population feedback functions on the environment and $G(E, I)$ is a bounded function. More formally, $G(E, I)$ should satisfy $-\infty < -C \leq G(E, I) \leq C < \infty$ for some positive real value C, such that the value $E_i = 0$ (the zero equilibrium, also referred to as the trivial or boundary equilibrium) indeed represents a regular equilibrium value of the ODE above. The function $G(E, I)$ then represents the per-capita rate of change of E_i and any non-zero (non-trivial or internal) equilibrium of E_i fulfills the condition $G(E, I) = 0$. To handle more easily the continuation of zero equilibrium values for this type of environment variables and to be able to detect transcritical bifurcation points (also referred to as branching points) between an equilibrium curve with $E_i = 0$ and a curve with $E_i \neq 0$, this type of environment variable has to be labeled as `"PERCAPITARATE"` in the vector `EnvironmentState` (see the code box below) and its equilibrium condition has to be specified by the per capita growth rate $G(E, I)$.

The second type of environment variable, indicated with the keyword `"GENERALODE"`, is one that follows dynamics described by an ordinary differential equation (ODE), but $E_i = 0$ is not a potential equilibrium value for this environment variable. The ODE describing the dynamics of such an environment variable $E_i(t)$ is then of the general form:

$$\frac{dE_i}{dt} = G(E, I)$$

in which E is the vector of environment variables, I is the vector of population feedback functions on the environment and $G(E, I) \neq 0$ when $E_i = 0$. Such an environment variable can never have a zero

equilibrium value and transcritical bifurcation points between an equilibrium curve with $E_i = 0$ and a curve with $E_i \neq 0$ do not occur either. All equilibrium values of E_i satisfy the condition $G(E, I) = 0$. This type of environment variable has to be labeled as `"GENERALODE"` in the vector `EnvironmentState` (see the code box below) and its equilibrium condition has to be specified by the function $G(E, I)$.

The last type of environment variable are variables that represent measures (weighted integrals) of the population distribution itself. More formally, environment variables that can be expressed as:

$$E_i(t) \;=\; I_i(t) \qquad \text{with} \qquad I_i(t) \;=\; \int_{\Omega} \gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, n(t, \boldsymbol{\chi})\, d\boldsymbol{\chi}$$

in which the function $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is some arbitrary weighing function and I_i is one of the functions representing the feedback of a population on its environment. This type of environment variable represents a direct density-dependent effect of the population on the life history of the individuals. Examples of the weighing functions include $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$, in which case E_i would represent the total population density in numbers, or $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = \boldsymbol{\chi}_i$ with $\boldsymbol{\chi}_i$ referring to the mass of an individual organism, in which case E_i would represent the total population biomass. Obviously, the value of E_i equals 0 in case of a zero-valued or trivial equilibrium state for the population distribution $n(t, \boldsymbol{\chi})$. This type of environment variable has to labeled as `"POPULATIONINTEGRAL"` in the vector `EnvironmentState` (see the code box below) and its equilibrium condition has to be specified by identifying it with the appropriate feedback function I_i.

In the PNAS model the variable `EnvironmentState` is defined as:

Code block 4.3.1.3

```
1 EnvironmentState <- c(R = "GENERALODE", P = "PERCAPITARATE", Bv = "POPULATIONINTEGRAL")
```

The first environment variable represents the resource density, which follows semi-chemostat growth in the absence of consumers. As a consequence, the resource R does NOT have an equilibrium value $\tilde{R} = 0$. The name of the first environment variable is therefore defined as `R` and its value is set to `"GENERALODE"`. The second environment variable represents the predator density, the dynamics of which is described by the ODE

$$\frac{dP}{dt} \;=\; \left(\epsilon \frac{aB}{1 + T_h B} \;-\; \delta \right) P$$

Because $P = 0$ is a regular fixed point of this ODE, the type of the second environment variable is set to `"PERCAPITARATE"` in the code box above. This environmental variable is given the name 'P'. Finally, the third environment variable in the PNAS model is the total biomass of small juvenile consumers B, which exerts a direct density-dependent effect on the mortality rate of small juvenile consumer individuals themselves, as it influences the predator functional response. The type of the third environment variable is therefore set to `"POPULATIONINTEGRAL"` and its name is defined as `Bv` in the code box above.

4.3.1.4 Default parameters

The last variable to be defined is the vector `DefaultParameters`, which should contain named vector elements that specify the name and default value of all parameters in the model:

Code block 4.3.1.4

```
1 DefaultParameters <- c(Rho = 0.1, Rmax = 3.0E-4, Lb = 7.0, Lv = 27.0, Lj = 110.0, Lm = 300.0, Beta = 9.0E-6,
      Imax = 1.0E-4, Rh = 1.5E-5, Gamma = 0.006, Rm = 0.003, Mub = 0.01, A = 5000.0, Th = 0.1, Epsilon = 0.5,
      Delta = 0.01)
```

The names of the vector elements (parameters) can be used in the programming of the life history functions of the model and are furthermore used to make the output files produced by the program more readable. These output files contain a small header text indicating among other details which parameter values were used for the computation of the results contained in the output file. In this report the parameter names are listed together with their value.

4.3.1.5 States at birth

The first function to be implemented for a particular life history model should be called `StateAtBirth()` and should define for every population in the model the actual value of the different individual state variables $\boldsymbol{\phi}_j = \left(\boldsymbol{\phi}_{j1}, \ldots, \boldsymbol{\phi}_{jk}\right)$ for every possible state-at-birth that an individual can be born with (i.e. the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$).

The function should return a vector with as many named vector elements as there are i-state variables. Each vector element should specify the name of the particular i-state variable and the numeric value with which the individual is born. The names of the vector elements can be used conveniently in the functions below that define the life history processes.

> If individuals can differ in their individual state at birth this function should return a matrix with the number of rows equal to the number of possible states at birth and the number of columns equal to the number of i-state variables. Each row then specifies the value of the individual state variable of the particular state at birth. In case the model accounts for multiple, structured populations this function should return a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables.
>
> In case the model accounts for multiple, structured populations and individuals can differ in their individual state at birth this function should return a 3-dimensional array with the first dimension having a length equal to the number of structured populations in the problem, the second dimension equal to the number of possible states at birth and the third dimension equal to the number of i-state variables.

For the PNAS model age at birth is (obviously) set to 0, while the length at birth is given by the parameter ℓ_b. The vector that is returned by the function `StateAtBirth()` hence consists of 2 elements named `Age` and `Length` with values 0 and ℓ_b, respectively:

Code block 4.3.1.5

```
StateAtBirth <- function(E, pars)
{
  with(as.list(c(E, pars)),{
        # We model a single structured population with two i-state variables:
        # 1: age (initial value 0); 2: length (initial value equal to parameter Lb)
        c(Age = 0.0, Length = Lb)
      })
}
```

4.3.1.6 Boundaries between consecutive stages

The next function `LifeStageEndings()` determines the boundaries between consecutive stages in the individual life history. It should return a variable named `maturation`, the value of which specifies the threshold value at which the current life stage of the individual ends and the individual matures to the next life history stage. The life stage that the individual is in at the moment this routine is called, is determined by the function argument `lifestage`, which has a value of 1 if the individual is in the first

life stage and a value equal to `PSPMdimensions["LifeHistoryStages"]` if it is in the last life stage. The end of the current life history stage, as indicated by `lifestage`, occurs when this threshold value becomes 0 and switches sign from negative to positive. For the end of the last life stage the death of old age, either by reaching the maximum age (`NumericalOptions["MAX_AGE"]`) or by reaching the minimum survival threshold (`NumericalOptions["MIN_SURVIVAL"]`), does not have to be specified separately, the program takes care of that automatically. For the final life stage hence return a constant negative value (for example, -1). In case the model accounts for multiple, structured populations the return variable `maturation` is a vector with the number of elements equal to the number of structured populations in the problem, while the argument `lifestage` is also a vector of a length equal to the number of structured populations in the model.

In the PNAS model the function `LifeStageEndings()` is defined as:

Code block 4.3.1.6

```
LifeStageEndings <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(E, pars, istate)),{
        maturation  = switch(lifestage, Length - Lv, Length - Lj, -1)
      })
}
```

In the model there is a discontinuous change at the length threshold $\ell = \ell_v$ when individuals turn from vulnerable to completely invulnerable to predation. The value that indicates the end of the first life stage (when `lifestage` equals 1) is hence set to $\ell - \ell_v$. Furthermore, individuals mature at $\ell = \ell_j$, which changes their fecundity discontinuously from a 0 value just before maturation to a positive value just after maturation. The value that indicates the end of the second (juvenile) stage (when `lifestage` equals 2) is hence set to $\ell - \ell_j$.

The threshold value returned to the program in `maturation` will in general depend on the individual state variables, possibly on the individual's state-at-birth and will be different for individuals in different life stages. For this reason, the function `LifeStageEndings()` has as arguments `lifestage`, specifying the life stage that the individual is currently in, `istate`, the individual state, and `birthstate`, the individual's state-at-birth. In addition, the threshold value marking the end of a particular stage may depend on the value of the environment variables contained in `E` and the parameter values contained in `pars`, which are hence also passed arguments to the function `LifeStageEndings()`.

> This routine will be called as many times as there are possible states-at-birth, because the state-at-birth may influence the threshold between consecutive life stages. The same holds for the next 2 routines discussed, which define changes in the *i-state* variables, the fecundity, mortality and the impact of individuals. In essence, individuals with different states-at-birth are treated as constituting subpopulations within the same structured population. Because of the possible dependence on the state-at-birth the variables `birthstate` and `BirthStateNr` are passed as arguments to the function `LifeStageEndings()` as well as to the 2 functions discussed below. These arguments contain the values of the *i-state* variables and the index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$, respectively, for which the routine is invoked and for which the threshold between consecutive life stages has to be evaluated.

4.3.1.7 Life history rates

The next function, named `LifeHistoryRates()`, specifies the life history rates of an individual. The function should return a list with 4 components, named `development`, `fecundity`, `mortality` and `impact`. The components should have the following structure:

- `development`: This component of the returned list specifies the right-hand side of the ODE:

$$\frac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}, \boldsymbol{\chi}_b)$$

determining the continuous development of the individual state variables during the life history. It should be a vector of length equal to the number of i-state variables. Each element specifies the rate of development for the particular i-state variable.

Notice that the development rate may differ in different life stages, for example growth in body size may be different for juveniles and adults in case adults invest a lot of energy into reproduction. The development rates should then be specified dependent on the current life stage the individual is in, which is determined by the function argument `lifestage`. The development rate may furthermore depend on the individual and environmental state variables and on the parameters, i.e. the values of the arguments `istate`, `E` and `pars`, respectively, but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate` and `BirthStateNr`, respectively.

In case the model accounts for multiple, structured populations this component should be a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables. In this case, the value of `development[p,i]` determines for each structured population `p` the development in the individual state variable `i`.

- `fecundity`: This component of the returned list specifies the current fecundity of the individual. In the most common case of a unique state-at-birth and a single structured population, like in the PNAS2002 model, the component `fecundity` should be a single value.

The fecundity will certainly depend on the life stage that the individual is in (only adults reproduce), which is contained in the function argument `lifestage`, on the individual and environmental state variables and on the parameters, i.e. the values of the arguments `istate`, `E` and `pars`, respectively, but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate` and `BirthStateNr`, respectively.

In case the model accounts for multiple, structured populations this component should be a matrix of fecundities with the number of rows equal to the number of structured populations in the problem and a single column. In case individuals can be born with different states at birth the component should have a number of columns equal to the number of states at birth. In this latter case not only the fecundity (i.e. the number of offspring produced per unit time) has to be specified, but also the state-at-birth of the produced offspring. Therefore, this function has to assign values to the matrix `fecundity[p,b]`, which determines for the population with index `p` the number of offspring produced per unit time with state-at-birth with index `b` in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$. Each column should hence specify the number of offspring produced with the particular state at birth.

- `mortality`: A single value specifying the current mortality rate that the individual experiences, possibly dependent on the life stage the individual is in at the moment this routine is called (given in the function argument `lifestage`), the current values of the individual and environmental state variables and parameters, i.e. the values of the arguments `istate`, `E` and `pars`, respectively, and the individual's state-at-birth (current values and index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$ given by `birthstate` and `BirthStateNr`, respectively).

In case the model accounts for multiple, structured populations this argument is a vector of mortality rates with the number of elements equal to the number of structured populations in the problem.

- `impact`: A single value or a vector of a length equal to the number of impact functions that need to be monitored for the individual. The value (or the values of the vector) should specify the current contribution of the individual to this population-level impact.

  As explained in section 4.3.1.1 interaction functions are of the form

  $$\int_{\Omega} h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, \tilde{n}(\boldsymbol{\chi})\, d\boldsymbol{\chi}$$

  where $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is an interaction (weighing) function that can depend on the values of the individual state $\boldsymbol{\chi}$, the state-at-birth of individuals $\boldsymbol{\chi}_b$ and the values of the environment variables E, and $\tilde{n}(\boldsymbol{\chi})$ is the stable population distribution in equilibrium. Such quantities can therefore represent the total population density in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$), the total population biomass (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ equals the biomass of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$) or the total population birth rate in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is the fecundity of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$). These interaction variables, in fact, determine the equilibrium of the model, because the nonlinearities that make an equilibrium possible arise through the impact of an individual on its environment.

  As also explained in section 4.3.1.1 all interaction functions are saved to the output file when an equilibrium has been computed. Interaction functions are hence not only used to compute the density-dependent feedback in the model, but also to obtain model output quantities of the form shown above.

  It should be pointed out that the element `impact` of the returned list should only specify the impact of an individual on its environment, given its current life stage (determined by the function argument `lifestage`), its individual state (argument `istate`), its state-at-birth (values and index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$ given by `birthstate` and `BirthStateNr`, respectively) and the value of environment variables `E` and parameters `pars`. In other words, the routine should only specify the weighing function $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$. The program automatically translates this individual-level impact function to the feedback of the total population on its environment.

  In case the model accounts for multiple, structured populations this component should be a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of impact functions.

For the PNAS2002 model the function `LifeHistoryRates()` is specified as follows:

Code block 4.3.1.7

```
LifeHistoryRates <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(E, pars, istate)),{
    list(
      # We model a single structured population (nrow=1) with two i-state variables:
      # 1: age (developmental rate 1.0); 2: Length (vonBertalanffy growth rate)
      development = c(1.0, Gamma*(Lm*R/(R + Rh) - Length)),
      fecundity   = switch(lifestage, 0, 0, Rm*R/(R + Rh)*Length^2),
      mortality   = switch(lifestage, Mub + A*P/(1+A*Th*Bv), Mub, Mub),
      impact      = switch(lifestage, c(Imax*R/(R + Rh)*Length^2, Beta*Length^3, 0, 0),
                                      c(Imax*R/(R + Rh)*Length^2, 0, Beta*Length^3, 0),
                                      c(Imax*R/(R + Rh)*Length^2, 0, 0, Beta*Length^3))
    )
  })
}
```

The first individual state variable in the model corresponds to the individual age, which obviously has a rate of development equal to 1. The rate of development in individual length, the second

individual state variable, follows the vonBertalanffy growth function, as specified by the function $g(\ell, R)$ (refer to the model formulation in section 4.2).

The code fragment above implements a non-zero fecundity for individuals in the third life stage (when `lifestage` equals 3), which corresponds to the adult individuals with $\ell > \ell_j$. The implemented expression corresponds to the function $\beta(\ell, R) = r_m R/(R_h + R)\ell^2$ as assumed in the PNAS model (see section 4.2).

In the PNAS model all individuals experience a background mortality rate μ_b, while small juvenile individuals, which are in the first distinguished life stage (when `lifestage` equals 1), experience on top of the background mortality a predation mortality equal to $aP/(1 + T_h B)$ as expressed by the function $\mu(\ell, P)$ in section 4.2.

In the PNAS model the impact of an individual consumer on its environment consists of its grazing rate $I(\ell, R)$ and the biomass-length relation $\omega\ell^3$ determining the biomass of vulnerable consumers, as it represents food for predators. The grazing rate of an individual consumer in the PNAS model is independent of the life stage it is in. As is shown in the code box above, this impact is assigned to the first interaction variable. The code box above furthermore shows that biomass of juvenile consumers that are vulnerable to predation is assigned to the second interaction variable, whereas the biomass of the invulnerable juvenile and adult consumers is assigned to the third and fourth interaction variable, respectively. These last two interaction variables are obviously not needed for the specification of the equilibrium, but are only included as additional output.

> Refer to the remarks in the discussion of the function `LifeStageEndings()` concerning the dependence on the individual's state-at-birth.

4.3.1.8 Optional discrete changes at stage boundaries

Even though not listed among the basic assumptions of the PSPM in the beginning of this chapter, it is permissible to have discrete changes or jumps in the individual state variables at the transition between two consecutive life stages. If these occur, they should be specified in the function `DiscreteChanges()`. This function is not relevant in case of the PNAS2002 model, in which case it can simply be left away or commented out.

Code block 4.3.1.8

```
DiscreteChanges <- function(lifestage, istate, birthstate, BirthStateNr, E, pars) {
  with(as.list(c(E, pars)),{
       # No discrete changes in this problem, function is commented out, which
       # would be equivalent to returning a copy of the input argument 'istate'
       istate
    })
}
```

If defined, the function `DiscreteChanges()` is called whenever a transition between two consecutive life stages is reached during the integration over the individual life history. The function should return a vector of length equal to the number of i-state variables. Each element should specify the value of the particular i-state variable after the transition to the current state. In case the model accounts for multiple, structured populations this function should return a matrix with the number of rows equal to the number of structured populations in the problem and the number of columns equal to the number of i-state variables.

It should be noted that the value of the variable `lifestage` indicates the life stage that is entered, that is, following the current stage boundary. This routine will hence never be called with a value of one of the elements `lifestage` equal to 1. The discrete changes in the individual state variables have to be implemented by assigning new values to the variables `istate`. These assignments may as before depend on the life stage that is entered, as specified by the variable `lifestage`, the (old

values) of the individual state variables, contained in the argument `istate`, and possibly on the individual's state-at-birth, specified in the argument `birthstate`. If no assignment of a value to `istate` is implemented, that particular individual state variable will keep its current value.

4.3.1.9 Equilibrium conditions for environmental state variables

The last function to be specified should be called `EnvEqui()` and should specify the equilibrium conditions of the environmental state variables as a function of the values of the population feedback functions, the values of the environment variables themselves, and the values of the model parameters. These values are passed as the arguments `I`, `E` and `pars` to the function. The function should return a vector with a length equal to the number of environmental state variables in the problem, in which each vector element specifies the equilibrium condition for the particular environmental state variable. Notice that the ordering of the elements in the returned vector should correspond to the ordering of the variables in `E`, meaning for example that the equilibrium condition for the second environment variable `E[2]` has to be returned in the second element of the return vector.

Notice that the feedback functions `I` are the population-level representations of the individual-level impact functions `impact` as they are defined in code box 4.3.1.7. In case of a unique state-at-birth the expected life history is the same for all individuals in the population and hence not dependent on a state-at-birth $\boldsymbol{\chi}_b$. In this case, such an expected impact of an individual during its entire life on its environment is given by an integral of the form:

$$\Gamma \;=\; \int_0^\infty \gamma(\boldsymbol{\chi}(a), E)\, \mathcal{F}(a)\, da$$

in which $\gamma(\boldsymbol{\chi}(a), E)$ quantifies the impact dependent on the individual state and the environment variables, $\boldsymbol{\chi}(a)$ is the individual state at age a and $\mathcal{F}(a)$ represents the probability that the individual survives until age a, which is defined as:

$$\mathcal{F}(a) \;=\; \exp\left(-\int_0^a \mu(\boldsymbol{\chi}(a), E)\, da\right)$$

with $\mu(\boldsymbol{\chi}(a), E)$ the individual's instantaneous mortality rate. The population-level impact on the environment now simply equals the product of the total population birth rate in equilibrium $\tilde{b}$ and the individual-level impact:

$$\tilde{I} = \tilde{b}\,\Gamma$$

The elements of the vector `I` and `impact` therefore correspond one-to-one, such that for example in the PNAS model, in which `impact[1]` is defined as the individual feeding rate on the resource, the quantity `I[1]` equals the grazing rate of the entire population on the resource.

In the PNAS model the function `EnvEqui()` is defined as:

Code block 4.3.1.9

```
EnvEqui <- function(I, E, pars) {
  with(as.list(c(E, pars)),{
        c(Rho*(Rmax - R) - I[1], Epsilon*A*I[2]/(1+A*Th*I[2]) - Delta, I[2])
      })
}
```

The first environment variable in the model represents the resource density, which follows semi-chemostat growth in the absence of consumers. As a consequence, the resource R does NOT have an equilibrium value $\tilde{R} = 0$. Its equilibrium condition is therefore specified in the code box above as `Rho*(Rmax - R) - I[1]`. The first term in this expression implements the semi-chemostat dynamics

in the absence of consumers and the quantity `I[1]` represents the grazing rate by the total population of consumers. These two quantities should cancel each other for the resource density to be in equilibrium.

The second environment variable represents the predator density, the dynamics of which is described by the ODE

$$\frac{dP}{dt} \;=\; \left(\epsilon \frac{aB}{1+T_h B} \;-\; \delta\right) P$$

Because $P = 0$ is a regular fixed point of this ODE, its per capita growth rate $\epsilon aB/(1+T_h B) - \delta$ is used to define its equilibrium condition. Notice in this respect that the population feedback quantity `I[2]` represents B, the total biomass of small juvenile consumers that are vulnerable to predation, which is calculated from the individual-level impact quantity `impact[2]` defined in code box 4.3.1.7.

Finally, the third environment variable in the PNAS model is the total biomass of small juvenile consumers B, which exerts a direct density-dependent effect on the mortality rate of small juvenile consumer individuals themselves, as it influences the predator functional response. The equilibrium condition of this third environment variable is specified by identifying it with the population feedback quantity `I[2]`, which the program computes on the basis of the individual-level impact quantity `impact[2]` that represents the biomass of an individual consumer that is in the first life stage, where it is vulnerable to predation (see the definition of `impact` in code box 4.3.1.7).

> In case of multiple states-at-birth, individuals with different states-at-birth may have different impacts on their environment. The expected impact of an individual during its entire life on its environment is then given by an integral of the form:
>
> $$\Gamma_j \;=\; \int_0^\infty \gamma(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)\,\mathcal{F}_j(a)\,da$$
>
> in which $\gamma(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)$ quantifies the impact of an individual that is born with state $\boldsymbol{\phi}_j$ state dependent on its individual state at age a, $\boldsymbol{\chi}(a,\boldsymbol{\phi}_j)$, its state-at-birth $\boldsymbol{\phi}_j$ and the environment variables. $\mathcal{F}_j(a)$ now represents the probability that an individual born with state-at-birth $\boldsymbol{\phi}_j$ survives until age a, which is defined as:
>
> $$\mathcal{F}_j(a) \;=\; \exp\left(-\int_0^a \mu(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)\,da\right)$$
>
> with $\mu(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)$ the individual's instantaneous mortality rate.
>
> If the possible states-at-birth are given by the set $\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_m\}$, the individual-level impact is an m-dimensional vector $\Gamma = (\Gamma_1,\ldots,\Gamma_m)$. The population-level impact on the environment in this case equals the dot product of this vector Γ with the m-dimensional vector $\tilde{b}$, representing the equilibrium distribution of produced offspring over the possible states-at-birth $\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_m\}$ (refer to Diekmann, Gyllenberg, and Metz (2003) for details).
>
> The program automatically computes the equilibrium distribution of produced offspring over the possible states-at-birth and uses it to compute the population-level impacts contained in `I` from the individual-level impacts that have been specified in `impact`.

4.3.2 Implementation of the model in `C`

The implementation of the model described in section 4.2 for analysis with the software package requires the specification of 12 pieces of `C`-code that can be subdivided into three different groups:

- Problem dimensions, numerical settings and model parameters

- Definition of the individual life history functions, such as development (growth), fecundity and mortality.
- Definition of the individual feedback on the environment and the equilibrium conditions for environment variables.

The pieces of C-code are discussed in detail in the next 12 subsections. The code can be found in the file PNAS2002.h, which can be opened by executing the command showpspm("PNAS2002.h"). To implement your own model you only need a basic understanding of C, which programming language I will not further discuss here. It is advisable to use one of the example models, which can be listed using the utility function showpspm(), as a basis for the implementation. To do so, you can copy the contents of the file PNAS2002.h to a new file in Rstudio's built-in editor and save this new file with a new name. The extension of your model-specific file should however remain '.h'. For ease of writing I will in the following sections often refer to this model as the PNAS model. The first 10 sections with C-code, specifying model constants and the individual life history, are to a considerable extent similar to the corresponding sections with code snippets discussed in chapter 3 on demographic analysis. Some of the text presented in that chapter is therefore repeated here for those readers that skipped the previous chapter.

The software allows for the analysis of models with multiple structured populations, each of which consists of individuals that are characterized by a finite number of individual state variables. The number of state variables characterizing an individual should, however, be the same for each of the structured populations in the model. Furthermore, at birth individuals may have one of a finite number of states-at-birth. To distinguish between populations, between individual state variables and between different states-at-birth, in the following sections the index p will consistently refer to the index of the structured population in the model. Because the dimension setting POPULATION_NR is used to specify the number of populations in the model (see the next section), p should have values in the range $0, 1, \ldots,$ POPULATION_NR-1. Similarly, the index i will consistently refer to the index of a particular individual state variable, which should always take values in the range $0, 1, \ldots,$ I_STATE_DIM-1, given that the dimension setting I_STATE_DIM determines the number of individual state variables (see the next section).

4.3.2.1 Definition of problem dimensions and numerical settings

The code box below defines the different dimensions of the model and the numerical settings to be used in the computations. These definitions, using #define-statement interpreted by the C-precompiler, have to appear at the very beginning of the model-specific file for the code to compile correctly.

The software can handle problems with multiple structured populations. Therefore, the variable POPULATION_NR has to be defined equal to the number of structured populations accounted for in the model. For the PNAS example this is obviously equal to 1 (line 2 in the code box below).

Code block 4.3.2.1

```
1 // Dimension settings: Required
  #define POPULATION_NR       1
  #define STAGES              3
  #define I_STATE_DIM         2
5 #define ENVIRON_DIM         3
  #define INTERACT_DIM        4
  #define PARAMETER_NR        16

  // Numerical settings: Optional (default values adopted otherwise)
10 #define MIN_SURVIVAL        1.0E-9          // Survival at which individual is considered dead
  #define MAX_AGE             100000          // Give some absolute maximum for individual age
```

```
#define DYTOL               1.0E-7          // Variable tolerance
#define RHSTOL              1.0E-6          // Function tolerance

#define ALLOWNEGATIVE       0               // Negative solution values allowed?
#define COHORT_NR           100             // Number of cohorts in state output
```

The variable `STAGES` has to be defined equal to the number of life stages that can be distinguished in the individual life history (line 3 in the code box above). While integrating the ODEs for the individual life history numerical problems may occur when the right hand side of the ODEs changes abruptly in value at a certain threshold value of the individual state, as a consequence of discontinuities in the development rate, the mortality rate or the fecundity. Each of such thresholds in the life history should be distinguished as a stage boundary. In the PNAS model the mortality changes discontinuously at $\ell = \ell_v$, while the fecundity changes discontinuously at $\ell = \ell_j$. Three life stages can hence be distinguished: vulnerable juveniles, invulnerable juveniles and adults, and the changes in the life history rates are indeed abrupt at the transition boundaries between these stages. The variable `STAGES` is therefore set equal to 3.

The variable `I_STATE_DIM` (line 4 in the code box above) defines the dimension of the individual state. For the PNAS model this is defined equal to 2 to account for both individual age and individual length.

The variable `ENVIRON_DIM` is required in nonlinear PSPM, whereas it is optional for demographic analysis of linear PSPMs. It represents the number of environment variables that determine the life history of an individual. In the PNAS model the growth and fecundity of individual consumers are functions of the resource density R, whereas the mortality is a function of the predator density P and the biomass of vulnerable consumers B. The latter only influences the mortality of the vulnerable consumers, because it determines the value of the predator functional response. `ENVIRON_DIM` hence equals 3 in the PNAS model.

The variable `INTERACT_DIM` defines the number of functions that represent the impact of an individual on its environment. In the PNAS model this feedback of an individual consumer on its environment consists of its grazing rate $I(\ell, R)$ and the biomass-length relation $\omega\ell^3$ determining the biomass of vulnerable consumers, as it represents food for predators. Therefore, the variable `INTERACT_DIM` should be at least set equal to 2. However, all interaction functions are also saved to the output file generated during a computation. The interaction functions can hence be conveniently used to produce arbitrary output quantities of the form

$$\int_\Omega h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, \tilde{n}(\boldsymbol{\chi})\, d\boldsymbol{\chi}$$

where $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is an interaction (weighing) function that can depend on the values of the individual state $\boldsymbol{\chi}$, the state-at-birth of individuals $\boldsymbol{\chi}_b$ and the values of the environment variables E, and $\tilde{n}(\boldsymbol{\chi})$ is the stable population distribution in equilibrium. Such quantities can therefore represent the total population density in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$), the total population biomass (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ equals the biomass of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$) or the total population birth rate in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is the fecundity of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$). In the PNAS model I want in addition to the biomass of vulnerable consumers, also the biomass of non-vulnerable juvenile consumers and the biomass of adult consumers as output of the model and hence have set the variable `INTERACT_DIM` equal to 4.

The last required parameter that has to be specified is the number of parameters in the model (set in line 7 in the code box above). In the PNAS model this equals 16 (ρ, R_{max}, ℓ_b, ℓ_v, ℓ_j, ℓ_m, ω, I_{max}, R_h, γ, r_m, μ_b, a, T_h, ϵ and δ).

The remaining definitions in the code box are all optional and can be left away. A list of all possible variables that can be changed by a definition in this code section is provided in section 9.3. The

variable MIN_SURVIVAL determines the threshold of the survival probability below which an individual is considered dead. The integration over the individual life history stops whenever the survival probability falls below this threshold value. In the code box above (line 10) the minimum survival is set to 10^{-9}, which is in fact the default value and is hence superfluous. Note that the value of MIN_SURVIVAL can *not* be set equal to 0. As an alternative to using 0 MIN_SURVIVAL can be set to a very small value like 10^{-100}.

The variable MAX_AGE (line 11 in code box above) can be used as an alternative to determine the end of an individual life and to stop the integration over the individual life history. In the PNAS model there is no maximum individual age and hence the variable is set to a very high value (100000), which the individuals will never reach, because before that age their survival probability has already dropped below its threshold value (10^{-9}).

The two quantities DYTOL and RHSTOL determine whether a solution has been found. In general, both demographic analysis as well as equilibrium analysis of PSPMs boils down to solving a system of nonlinear equations that can be represented as $G(y) = 0$ for a set of unknowns y in iterative manner. The subsequent estimates of the solution in the Newton iterations can be labeled as y_p and y_{p+1}. A solution is now considered to be located if both of the following conditions hold:

$$\|y_{p+1} - y_p\| < \epsilon_y$$

$$\|G(y_{p+1})\| < \epsilon_G$$

where $\|.\|$ refers to the Euclidean norm. DYTOL and RHSTOL are the quantities ϵ_y and ϵ_G, respectively. Increasing (decreasing) their value leads to easier (harder) acceptance of a set of unknowns as a solution to the system of equations $G(y) = 0$. The definition of these two accuracies in the code box is in fact superfluous as they are defined equal to their default values (see section 9.3).

The quantity ALLOWNEGATIVE is a flag that can only have a value of 0 or 1 and determines whether or not computations should stop when one of the variables to solve for reaches a negative value. In most population models negative solution values are biologically not relevant and ALLOWNEGATIVE is hence set to 0 by default. Line 16 in the code box above is only included to illustrate the use of ALLOWNEGATIVE and does not change the value of this variable from its default value. Most likely, setting ALLOWNEGATIVE equal to 1 as opposed to 0 will only be useful in specific cases.

The last quantity COHORT_NR defines the number of cohorts making up the equilibrium population output. During computations of the equilibrium a number of output files will be generated (see section 4.4.5), one of which is a file containing the population equilibrium state for each parameter that the equilibrium values are computed for. COHORT_NR specifies how many cohorts should be used to represent these equilibrium population states. Larger values will generate more detailed representations of the equilibrium population state at the expense of larger file sizes.

4.3.2.2 Definition of parameter names and values

The code box below assigns each of the model parameters a meaningful name and a default value.

Code block 4.3.2.2

```
// Descriptive names of parameters in parameter array (at least two parameters are required)
char  *parameternames[PARAMETER_NR] =
    { "Rho", "Rmax", "Lb", "Lv", "Lj", "Lm", "Beta", "Imax", "Rh", "Gamma", "Rm", "Mub",
      "A", "Th", "Epsilon", "Delta"};

// Default values of all parameters
double  parameter[PARAMETER_NR] =
    { 0.1, 3.0E-4, 7.0, 27.0, 110.0, 300.0, 9.0E-6, 1.0E-4, 1.5E-5, 0.006, 0.003, 0.01,
      5000.0, 0.1, 0.5, 0.01};
```

Model parameter values are stored by the program in the vector variable `parameter`. The lines 2-4 above assign each of the elements this vector a more meaningful, model-specific name. These name strings can not be used in the remaining parts of the model implementation, they only serve to make the output files produced by the program more readable. These output files contain a small header text indicating among other details which parameter values were used for the computation of the results contained in the output file (see section 4.4.5). In the output file parameters are referred to with their names as defined in the array of strings `*parameternames[`k`]`. To adapt the above code to a different model, the code on line 2 of the code box above should remain the same, only change lines 3-4 as needed (possibly extending it over more lines in case there are many parameters).

The default values to use for the model parameters are specified by the declaration of the vector `parameter[PARAMETR_NR]` on line 7-9 of the previous code box. The values should be specified as a comma-separated array of values within braces (don't forget the closing semi-colon at the end of the statement that is required in the `C`-language!). To adapt the above code to a different model, the code on line 7 of the code box above should remain the same, only change lines 8-9 as needed (possibly extending it over more lines in case there are many parameters).

4.3.2.3 Definition of aliases to simplify implementation

The following code box defines aliases for program variables used in the `C`-implementation of the model, such that they are more easily identified with the model ingredients. Defining these aliases is optional but strongly advised as it makes model implementation more straightforward.

The life history functions in any model depend on the individual state itself, on the environment variables and on model parameters. The value of the individual state variables at a particular age are always referred to with the program variable `istate[`p`][`i`]`, where the index p refers to the number of the population and the index i refers to the number of the individual state variables. Notice that in `C` array indices run from 0 (as opposed to 1 like in `R`)! Similarly, the value of the individual's state variables at birth are always referred to with the program variable `birthstate[p][i]`. In case there are multiple populations and/or more than a single individual state variable, it is up to the user to keep track of which index pertains to which population or individual state variable. In the PNAS model the two individual state variables are age and length, which are identified with the first and second element of the individual state vector, `istate[0][0]` and `istate[0][1]`, respectively (line 2-3 in the code box below).

Code block 4.3.2.3

```
1  // Aliases definitions for all istate variables
   #define AGE                 istate[0][0]
   #define LENGTH              istate[0][1]

5  // Aliases definitions for all environment variables
   #define R                   E[0]
   #define P                   E[1]
   #define B                   E[2]

10 // Aliases definitions for all parameters
   #define RHO                 parameter[ 0]   // Default: 0.1
   #define RMAX                parameter[ 1]   // Default: 3.0E-4

   #define LB                  parameter[ 2]   // Default: 7
15 #define LV                  parameter[ 3]   // Default: 27
   #define LJ                  parameter[ 4]   // Default: 110
   #define LM                  parameter[ 5]   // Default: 300

   #define OMEGA               parameter[ 6]   // Default: 9.0E-6
20
   #define IMAX                parameter[ 7]   // Default: 1.0E-4
```

```
#define RH                  parameter[ 8]   // Default: 1.5E-5

#define GAMMA               parameter[ 9]   // Default: 0.006
#define RM                  parameter[10]   // Default: 0.003

#define MUB                 parameter[11]   // Default: 0.01

#define A                   parameter[12]   // Default: 5000.0
#define TH                  parameter[13]   // Default: 0.1
#define EPSILON             parameter[14]   // Default: 0.5
#define DELTA               parameter[15]   // Default: 0.01
```

The value of the environment variables are contained in an array `E[`e`]` with e an index in the range 0 ... `ENVIRON_DIM-1`. Again, it is up to the user to keep track of which index pertains to which environmental state variable. The use of aliases is really beneficial for this purpose. As defined in code box 4.3.2.1 three environment variables are identified in the PNAS model: the resource density, the density of predators and the biomass of juvenile consumers that are vulnerable to predation. Lines 6-8 in the code box above identifies these with the first, second and third element of the array `E[`e`]`, respectively, and introduces the aliases `R`, `P` and `B` for these quantities. All code shown below will make use of these aliases as opposed to their real names in the program (`E[0]`, `E[1]` and `E[2]`).

Similar arguments as given above for the individual state variables contained in the array `istate[`p`][`i`]` and the environment variables contained in the array `E[`e`]` hold for the model parameters. All model parameters are contained in a vector named `parameter[`k`]` in the code (see the previous section) with k an index in the range 0 ... `PARAMETER_NR-1`. Which element of this vector represents which model-specific parameter is up to the user. To prevent mixing up the interpretation of the different vector elements and hence to prevent mistakes, it is strongly advised to define meaningful, model-specific aliases for each of the elements of the vector `parameter[`k`]` as is illustrated in lines 11-32 in the code box above. It is best to avoid completely the direct use of the program variable `parameter[`k`]` in any part of the model specification and only use the models-specific aliases.

> As can be seen in the code block above all aliases for program variables used in the `C`-implementation of the model are names in capitals. It is advisable to use only capitals when introducing these aliases (or global variables if they are needed) to avoid any conflict between these aliases and variables that defined elsewhere in any of the `C` files with numerical routines that are included in the package.

4.3.2.4 Specifying the number of possible states-at-birth

The first routine to be implemented for a particular life history model defines for every population in the model the number of possible states-at-birth that an individual can be born with (i.e. the value of the size m of the set $\{\phi_1, \ldots, \phi_m\}$).

Code block 4.3.2.4

```
/*
 * Specify the number of states at birth for the individuals in all structured
 * populations in the problem in the vector BirthStates[].
 */

void SetBirthStates(int BirthStates[POPULATION_NR], double E[])
{
  BirthStates[0] = 1;

  return;
}
```

For each population with index p the variable `BirthStates[p]` has to be set to the number of possible states at birth. Because in the PNAS model all individuals are born with age 0 and length $\ell = \ell_b$, all individuals have the same, unique state at birth and hence `BirthStates[0]` is set to 1.

> Note that different populations may have different numbers of states-at-birth. `BirthStates[p]` hence does not need to be the same for all p.

4.3.2.5 Specifying the value of all possible states-at-birth

The next routine to implement defines for every possible state-at-birth with index j the actual value of the different individual state variables at birth $\boldsymbol{\phi}_j = \left(\boldsymbol{\phi}_{j1}, \ldots, \boldsymbol{\phi}_{jk}\right)$. %

Code block 4.3.2.5

```
1  /*
   * Specify all the possible states at birth for all individuals in all
   * structured populations in the problem. BirthStateNr represents the index of
   * the state of birth to be specified. Each state at birth should be a single,
5  * constant value for each i-state variable.
   *
   * Notice that the first index of the variable 'istate[][]' refers to the
   * number of the structured population, the second index refers to the
   * number of the individual state variable. The interpretation of the latter
10 * is up to the user.
   */

  void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
  {
15  AGE = 0.0;
    LENGTH = LB;

    return;
  }
```

For every population ($p = 0, 1, \ldots,$ `POPULATION_NR`-1) the value of each individual state variable with index i, `istate[p][i]` ($i = 0, 1, \ldots,$ `I_STATE_DIM`-1), has to be assigned a unique value, from which individual development will start at age 0. Notice that the program does not automatically include individual age in its characterization of the individual state, even though integration over the entire life history (as a function of age) is carried out. For the PNAS model age at birth is (obviously) set to 0, while the length at birth is given by the parameter ℓ_b (line 15 and 16, respectively in the code box below).

> This routine will be called as many times as there are possible states-at-birth. The variable `BirthStateNr` indicates the index j of the state-at-birth in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$ for which the values have to be set in the current invocation of the routine. The routine will thus be called with `BirthStateNr` set equal to a value in $0, 1, \ldots, m-1$ (Remember the starting index 0 in C!). If there are multiple states-at-birth (`BirthStates[p]` > 1) the definition of the values of the *i-state* variables has to depend explicitly on the index `BirthStateNr` to make the states-at-birth different from each other. Furthermore, if the problem involves multiple structured populations the number of possible states-at-birth can be different for each of them, which might lead to a situation that the routine above is called with a value of the index `BirthStateNr` that is larger than the maximum number of states-at-birth for a particular population (`BirthStateNr` $\geq$ `BirthStates[p]`). The program safely ignores such inappropriate state-at-birth specifications.

4.3.2.6 Definition of boundaries between discrete stages

The next routine determines the boundaries between consecutive stages in the individual life history:

Code block 4.3.2.6

```
1  /*
   * Specify the threshold determining the end point of each discrete life
   * stage in individual life history as function of the i-state variables and
   * the individual's state at birth for all populations in every life stage.
5  *
   * Notice that the first index of the variable 'istate[][]' refers to the
   * number of the structured population, the second index refers to the
   * number of the individual state variable. The interpretation of the latter
   * is up to the user.
10 */

   void IntervalLimit(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                      double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                      double limit[POPULATION_NR])
15 {
     switch (lifestage[0])
     {
       case 0:
         limit[0] = LENGTH - LV;
20       break;
       case 1:
         limit[0] = LENGTH - LJ;
         break;
     }
25
     return;
   }
```

In this routine the variable `limit[`p`]` has to be defined, which has as many elements as there are populations ($p = 0 \ldots$ `POPULATION_NR-1`). The life stage that the individual is in at the moment this routine is called, is determined by the variable `lifestage[`p`]`, which has a value of 0 if the individual is in the first life stage and a value of `STAGES`-1 if it is in the last life stage. The element `limit[`p`]` should now indicate when the current life stage as given in `lifestage[`p`]` ends. In particular, the program considers the current life stage to end when `limit[`p`]` turns from negative to positive. For the end of the last life stage the death of old age, either by reaching the maximum age `MAX_AGE` or by reaching the minimum survival threshold `MIN_SURVIVAL`, does not have to be specified separately, the program takes care of that automatically.

The threshold value that has to be stored and returned to the program in `limit[p]` will depend on the individual state variables, possibly on the individual's state-at-birth and will be different for individuals in different life stages. For this reason, the routine `IntervalLimit()` has as arguments `lifestage[]`, specifying the life stage that the individual is currently in, `istate[][]`, the individual state, `birthstate[][]` and `BirthStateNr`, the value and index of the individual's state-at-birth, respectively. The threshold value marking the end of a particular stage may, however, in addition depend on the value of the environment variables (`E[]`).

In the PNAS model there is a discontinuous change at the length threshold $\ell = \ell_v$ when individuals turn from vulnerable to completely invulnerable to predation. The value that indicates the end of the first life stage (when `lifestage[`p`]` $= 0$) is hence set to $\ell - \ell_v$ (line 18-20 in the code box above). Furthermore, individuals mature at $\ell = \ell_j$, which changes their fecundity discontinuously from a 0 value just before maturation to a positive value just after maturation. The value that indicates the end of the second (juvenile) stage (when `lifestage[`p`]` $= 1$) is hence set to $\ell - \ell_j$ (line 21-23 in the code box above).

Like the previous routine, this routine will be called as many times as there are possible states-at-birth, because the state-at-birth may influence the threshold between consecutive life stages. The same holds for the routines discussed in sections 4.3.2.7-4.3.2.11 below, which define changes in the *i-state* variables, the fecundity and the mortality of individuals and their impact on the environment, respectively. In essence, individuals with different states-at-birth are treated as subpopulations within the same structured population. Because of the possible dependence on the state-at-birth the variables `birthstate[][]` and `BirthStateNr` are passed as arguments to this routine and the once discussed in sections 4.3.2.7-4.3.2.11. These arguments contain the values of the *i-state* variables and the index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$, respectively, for which the routine is invoked and for which the threshold between consecutive life stages has to be evaluated.

If the problem involves multiple structured populations and the number of possible states-at-birth differs among them, the routine above may be called with a value of the index `BirthStateNr` that is larger than the maximum number of states-at-birth for a particular population (`BirthStateNr` $\geq$ `BirthStates[`p`]`). Although this circumstance may seem confusing, the user does not have to worry about it, as the program is designed to safely ignore such assignments of thresholds between consecutive life stages, changes in the *i-state* variables, fecundity, mortality and impact on the environment of individuals for states-at-birth with index `BirthStateNr` $\geq$ `BirthStates[`p`]`that are inappropriate for the structured population with index p.

4.3.2.7 Specification of continuous individual state development

This routine specifies the right-hand side of the ODE:

$$\frac{d\boldsymbol{\chi}}{da} \;=\; g(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$$

that determines the continuous development of the individual state variables during the life history as a function of the state variables themselves, the individual's state-at-birth and the environment variables.

Code block 4.3.2.7

```
/*
 * Specify the development of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * Notice that the first index of the variables 'istate[][]' and 'development[][]'
 * refers to the number of the structured population, the second index refers
 * to the number of the individual state variable. The interpretation of the
 * latter is up to the user.
 */

void Development(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                 double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                 double development[POPULATION_NR][I_STATE_DIM])
{
  development[0][0] = 1.0;
  development[0][1] = GAMMA*(LM*R/(R + RH) - LENGTH);

  return;
}
```

For each individual state variable i of every structured population p that is part of the individual state `istate[`p`][`i`]`, its rate of development during the life history has to be specified in `development[`p`][`i`]`. Notice that these development rates may differ in different life stages, for example growth in body size may be different for juveniles and adults in case adults invest a lot of energy into reproduction. The development rates should then be specified dependent on the current life stage the individual is in. This current life stage at the moment the routine is evaluated is contained in the variable `lifestage[`p`]`. The development rate may furthermore depend on the individual state variables, on the individual's state-at-birth and on the value of the environment variables, which is the reason for `istate[][]`, the individual state, `birthstate[][]` and `BirthStateNr`, the value and index of the individual's state-at-birth, respectively, and `E[]`, the environment variables, as arguments to this routine.

In the PNAS model the first individual state variable corresponds to the individual age, which obviously has a rate of development equal to 1. The rate of development in individual length, the second individual state variable, follows the vonBertalanffy growth function, as specified by the function $g(\ell, R)$ (refer to the model formulation at the start of this chapter).

> Refer to the remarks in section 4.3.2.6 concerning the dependence on the individual's state-at-birth.

4.3.2.8 Specification of discrete individual changes at stage transitions

Even though not listed among the basic assumptions of the PSPM in the beginning of this chapter, it is permissible to have discrete changes or jumps in the individual state variables at the transition between two consecutive life stages. If these occur, they should be programmed in the following routine.

Code block 4.3.2.8

```
/*
 * Specify the possible discrete changes (jumps) in the individual state
 * variables when ENTERING the stage specified by 'lifestage[]'.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void DiscreteChanges(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                     double *birthstate[POPULATION_NR], int BirthStateNr, double E[])
{
  return;
}
```

This routine is not relevant in case of the PNAS model and hence its contents are empty (apart for the necessary `return;` statement).

This routine is called whenever a transition between two consecutive life stages is reached during the integration over the individual life history. It should be noted that the value of the variable `lifestage[`p`]` indicates the life stage that is entered, that is, following the current stage boundary. This routine will hence never be called with a value of one of the elements `lifestage[`p`]` equal to 0. The discrete changes in the individual state variables have to be implemented by assigning new values to the variables `istate[`p`][`i`]`. These assignments may as before depend on the life stage that is entered, as specified by the variable `lifestage[]`, the (old values) of the individual state variables, contained in the argument `istate[][]`, the individual's state-at-birth, determined by the arguments `birthstate[][]` and `BirthStateNr`, and on the environment variables `E[]`. If no assignment of a value to `istate[`p`][`i`]` is implemented, that particular individual state variable will keep its current value.

Refer to the remarks in section 4.3.2.6 concerning the dependence on the individual's state-at-birth.

4.3.2.9 Specification of fecundity

The following routine specifies the fecundity as a function of the individual state.

Code block 4.3.2.9

```
/*
 * Specify the fecundity of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * The number of offspring produced has to be specified for every possible
 * state at birth in the variable 'fecundity[][]'. The first index of this
 * variable refers to the number of the structured population, the second
 * index refers to the number of the birth state.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double *fecundity[POPULATION_NR])
{
  fecundity[0][0] = 0.0;
  if (lifestage[0] == 2)
    fecundity[0][0] = RM*R/(R + RH)*LENGTH*LENGTH;

  return;
}
```

In this routine not only the fecundity (i.e. the number of offspring produced per unit time) has to be specified, but also the state-at-birth of the produced offspring. Therefore, this routine has to assign values to the matrix `fecundity[`p`][`j`]`, which determines for the population with index p the number of offspring produced per unit time with state-at-birth with index j in the set $\{\phi_1, \ldots, \phi_m\}$. This fecundity will certainly depend on the life stage that the individual is in (only adults reproduce), which is contained in the argument `lifestage[]`, and on the individual state variables, i.e. the values of the argument `istate[][]`), but possibly also on the individual's state-at-birth, the values and index of which are specified by the arguments `birthstate[][]` and `BirthStateNr`, respectively, and on the value of environment variables, provided by the argument `E[]`, at the moment this routine is called.

In the most common case of a unique state-at-birth and a single structured population, like in the PNAS model, the only valid indices are $p = 0$ and $j = 0$ and hence only the variable `fecundity[0][0]` has to be assigned. The code fragment above implements a non-zero fecundity for individuals in the third life stage (`lifestage[0] == 2`), which corresponds to the adult individuals with $\ell > \ell_j$. The implemented expression corresponds to the function $\beta(\ell, R) = r_m R/(R_h + R)\ell^2$ as assumed in the PNAS model (see section 4.2). The code provides a good example of how to assign a different value for a particular life history rate dependent on the life stage that an individual is in. The same approach can also be used in the other routines specifying the life history rates of individuals.

For more detailed remarks about models with multiple states-at-birth consult section 4.3.2.6.

4.3.2.10 Specification of mortality

The following routine specifies the mortality as a function of the individual state.

Code block 4.3.2.10

```
/*
 * Specify the mortality of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 */

void Mortality(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double mortality[POPULATION_NR])
{
  if (lifestage[0] == 0)
    mortality[0] = MUB + A*P/(1+A*TH*B);
  else
    mortality[0] = MUB;

  return;
}
```

For each population the corresponding element of the array `mortality[`p`]` should be assigned the mortality rate, possibly dependent on the life stage the individual is in at the moment this routine is called (given in argument `lifestage[]`), the current *i-state* of the individual (given in argument `istate[][]`), the individual's state-at-birth (values and index in the set $\{\phi_1, \ldots, \phi_m\}$ given by `birthstate[][]` and `BirthStateNr`, respectively) and the value of environment variables (`E[]`).

In the PNAS model all individuals experience a background mortality rate μ_b, while small juvenile individuals, which are in the first distinguished life stage (`lifestage[0] == 0`), experience on top of the background mortality a predation mortality equal to $aP/(1 + T_h B)$ as expressed by the function $\mu(\ell, P)$ in section 4.2.

> Refer to the remarks in section 4.3.2.6 concerning the dependence on the individual's state-at-birth.

4.3.2.11 Specification of feedback impact on the environment

In this routine the functions should be programmed that represent the influence of individuals in the structured populations on their environment. These functions may represent effects like grazing rates or availability as food for higher trophic levels.

Code block 4.3.2.11

```
/*
 * For all the integrals (measures) that occur in interactions of the
 * structured populations with their environments and for all the integrals
 * that should be computed for output purposes (e.g. total juvenile or adult
 * biomass), specify appropriate weighing function dependent on the i-state
 * variables, the individual's state at birth, the environment variables and
 * the current life stage of the individuals. These weighing functions should
 * be specified for all structured populations in the problem. The number of
 * weighing functions is the same for all of them.
 *
 * Notice that the first index of the variables 'istate[][]' and 'impact[][]'
```

```
 * refers to the number of the structured population, the second index of the
 * variable 'istate[][]' refers to the number of the individual state variable,
 * while the second index of the variable 'impact[][]' refers to the number of
 * the interaction variable. The interpretation of these second indices is up
 * to the user.
 */

void Impact(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
            double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
            double impact[POPULATION_NR][INTERACT_DIM])
{
  impact[0][0] = IMAX*R/(R + RH)*LENGTH*LENGTH;

  switch (lifestage[0])
    {
    case 0:
      impact[0][1] = OMEGA*LENGTH*LENGTH*LENGTH;
      impact[0][2] = 0;
      impact[0][3] = 0;
      break;
    case 1:
      impact[0][1] = 0;
      impact[0][2] = OMEGA*LENGTH*LENGTH*LENGTH;
      impact[0][3] = 0;
      break;
    case 2:
      impact[0][1] = 0;
      impact[0][2] = 0;
      impact[0][3] = OMEGA*LENGTH*LENGTH*LENGTH;
      break;
    }

  return;
}
```

As explained in section 4.3.2.1 interaction functions are of the form

$$\int_\Omega h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, \tilde{n}(\boldsymbol{\chi})\, d\boldsymbol{\chi}$$

where $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is an interaction (weighing) function that can depend on the values of the individual state $\boldsymbol{\chi}$, the state-at-birth of individuals $\boldsymbol{\chi}_b$ and the values of the environment variables E, and $\tilde{n}(\boldsymbol{\chi})$ is the stable population distribution in equilibrium. Such quantities can therefore represent the total population density in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$), the total population biomass (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ equals the biomass of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$) or the total population birth rate in equilibrium (when $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is the fecundity of an individual with individual state $\boldsymbol{\chi}$ and state-at-birth $\boldsymbol{\chi}_b$). These interaction variables, in fact, determine the equilibrium of the model, because the nonlinearities that make an equilibrium possible arise through the impact of an individual on its environment.

As also explained in section 4.3.2.1 all interaction functions are saved to the output file when an equilibrium has been computed. Interaction functions are hence not only used to compute the density-dependent feedback in the model, but also to obtain model output quantities of the form shown above.

In the PNAS model this feedback of an individual consumer on its environment consists of its grazing rate $I(\ell, R)$ and the biomass-length relation $\omega\ell^3$ determining the biomass of vulnerable consumers, as it represents food for predators. The grazing rate of an individual consumer in the PNAS model is independent of the life stage it is in. As is shown in line 23 of the code box above, this impact is assigned to the first interaction variable (the one with index 0). Line 25-42 of the code box above show that biomass of juvenile consumers that are vulnerable to predation is assigned to the second

interaction variable (lines 27-31), whereas the biomass of the invulnerable juvenile and adult consumers is assigned to the third (lines 32-36) and fourth (lines 37-41) interaction variable. These last two interaction variables are obviously not needed for the specification of the equilibrium, but are only included as additional output.

It should be pointed out that the routine `Impact()` should only specify the impact of an individual on its environment, given its current life stage (function argument `lifestage[]`), its individual state (argument `istate[][]`), its state-at-birth (values and index in the set $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m\}$ given by `birthstate[][]` and `BirthStateNr`, respectively) and the value of environment variables (`E[]`). In other words, the routine should only specify the weighing function $h(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$. The program automatically translates this individual-level impact function to the feedback of the total population on its environment, as explained in the next section.

> Refer to the remarks in section 4.3.2.6 concerning the dependence on the individual's state-at-birth.

4.3.2.12 Specification of equilibrium conditions of the environment

The last routine has to specify the equilibrium conditions of the environment, dependent on the values of the environment variables itself and/or the values of the population feedback functions. As explained in section 4.1 environment variables can be of different types. As shown in the code box below 3 different types of environment variables are distinguished that are referred to with the keywords `PERCAPITARATE`, `GENERALODE` and `POPULATIONINTEGRAL`, respectively.

The first type of environment variable, indicated with the keyword `PERCAPITARATE`, is one that follows dynamics described by an ordinary differential equation (ODE) and in addition can potentially be 0 in equilibrium. The ODE describing the dynamics of such an environment variable $E_i(t)$ is then of the general form:

$$\frac{dE_i}{dt} \;=\; G(E, I)\, E_i$$

in which E is the vector of environment variables, I is the vector of population feedback functions on the environment and $G(E, I)$ is a bounded function. More formally, $G(E, I)$ should satisfy $-\infty < -C \leq G(E, I) \leq C < \infty$ for some positive real value C, such that the value $E_i = 0$ (the zero equilibrium, also referred to as the trivial or boundary equilibrium) indeed represents a regular equilibrium value of the ODE above. The function $G(E, I)$ then represents the per-capita rate of change of E_i and any non-zero (non-trivial or internal) equilibrium of E_i fulfills the condition $G(E, I) = 0$. To handle more easily the continuation of zero equilibrium values for this type of environment variables and to be able to detect transcritical bifurcation points (also referred to as branching points) between an equilibrium curve with $E_i = 0$ and a curve with $E_i \neq 0$, this type of environment variable has to be labeled as `PERCAPITARATE` in the program and its equilibrium condition has to be specified by the per capita growth rate $G(E, I)$.

The second type of environment variable, indicated with the keyword `GENERALODE`, is one that follows dynamics described by an ordinary differential equation (ODE), but $E_i = 0$ is not a potential equilibrium value for this environment variable. The ODE describing the dynamics of such an environment variable $E_i(t)$ is then of the general form:

$$\frac{dE_i}{dt} \;=\; G(E, I)$$

in which E is the vector of environment variables, I is the vector of population feedback functions on the environment and $G(E, I) \neq 0$ when $E_i = 0$. Such an environment variable can never have a zero

equilibrium value and transcritical bifurcation points between an equilibrium curve with $E_i = 0$ and a curve with $E_i \neq 0$ do not occur either. All equilibrium values of E_i satisfy the condition $G(E, I) = 0$. This type of environment variable has to be labeled as `GENERALODE` in the program and its equilibrium condition has to be specified by the function $G(E, I)$.

The last type of environment variable are variables that represent measures (weighted integrals) of the population distribution itself. More formally, environment variables that can be expressed as:

$$E_i(t) \;=\; I_i(t) \qquad \text{with} \qquad I_i(t) \;=\; \int_\Omega \gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)\, n(t, \boldsymbol{\chi})\, d\boldsymbol{\chi}$$

in which the function $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$ is some arbitrary weighing function and I_i is one of the functions representing the feedback of a population on its environment. This type of environment variable represents a direct density-dependent effect of the population on the life history of the individuals. Examples of the weighing functions include $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = 1$, in which case E_i would represent the total population density in numbers, or $\gamma_i(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E) = \boldsymbol{\chi}_i$ with $\boldsymbol{\chi}_i$ referring to the mass of an individual organism, in which case E_i would represent the total population biomass. Obviously, the value of E_i equals 0 in case of a zero-valued or trivial equilibrium state for the population distribution $n(t, \boldsymbol{\chi})$. This type of environment variable has to labeled as `POPULATIONINTEGRAL` in the program and its equilibrium condition has to be specified by identifying it with the appropriate feedback function I_i.

Code block 4.3.2.12

```
/*
 * Specify the type of each of the environment variables by setting
 * the entries in EnvironmentType[ENVIRON_DIM] to PERCAPITARATE, GENERALODE
 * or POPULATIONINTEGRAL based on the classification below:
 *
 * Set an entry to PERCAPITARATE if the dynamics of E[j] follow an ODE and 0
 * is a possible equilibrium state of E[j]. The ODE is then of the form
 * dE[j]/dt = P(E,I)*E[j], with P(E,I) the per capita growth rate of E[j].
 * Specify the equilibrium condition as condition[j] = P(E,I), do not include
 * the multiplication with E[j] to allow for detecting and continuing the
 * transcritical bifurcation between the trivial and non-trivial equilibrium.
 *
 * Set an entry to GENERALODE if the dynamics of E[j] follow an ODE and 0 is
 * NOT an equilibrium state of E. The ODE then has a form dE[j]/dt = G(E,I).
 * Specify the equilibrium condition as condition[j] = G(E,I).
 *
 * Set an entry to POPULATIONINTEGRAL if E[j] is a (weighted) integral of the
 * population distribution, representing for example the total population
 * biomass. E[j] then can be expressed as E[j] = I[p][i]. Specify the
 * equilibrium condition in this case as condition[j] = I[p][i].
 *
 * Notice that the first index of the variable 'I[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the interaction variable. The interpretation of the latter
 * is up to the user. Also notice that the variable 'condition[j]' should
 * specify the equilibrium condition of environment variable 'E[j]'.
 */

const int EnvironmentType[ENVIRON_DIM] = {GENERALODE, PERCAPITARATE, POPULATIONINTEGRAL};

void EnvEqui(double E[], double I[POPULATION_NR][INTERACT_DIM],
             double condition[ENVIRON_DIM])
{
  condition[0] = RHO*(RMAX - R) - I[0][0];
  condition[1] = EPSILON*A*I[0][1]/(1+A*TH*I[0][1]) - DELTA;
  condition[2] = I[0][1];

  return;
}
```

In the code box above, the array `EnvironmentType[ENVIRON_DIM]` has to define for each environment variable separately its type (`PERCAPITARATE`, `GENERALODE` or `POPULATIONINTEGRAL`). This array hence has as many elements as there are environment variables. Secondly, in the routine that follows the specification of `EnvironmentType[ENVIRON_DIM]` the equilibrium conditions have to be implemented as a function of the values of the environment variables itself `E[]` and the value of the population feedback functions `I[][]`. These latter two arrays are passed as arguments to the routine `EnvEqui`, while the equilibrium conditions have to be specified in the array `condition[]`. Notice that the ordering of the elements in the arrays `condition[]` and `E[]` are the same, meaning for example that the equilibrium condition for the second environment variable `E[1]` has to be returned in `condition[1]`.

Notice that the feedback functions `I[][]` are the population-level representations of the individual-level impact functions `impact[][]` as they are defined in section 4.3.2.11. In case of a unique state-at-birth the expected life history is the same for all individuals in the population and hence not dependent on a state-at-birth $\boldsymbol{\chi}_b$. In this case, such an expected impact of an individual during its entire life on its environment is given by an integral of the form:

$$\Gamma \;=\; \int_0^\infty \gamma(\boldsymbol{\chi}(a), E)\, \mathcal{F}(a)\, da$$

in which $\gamma(\boldsymbol{\chi}(a), E)$ quantifies the impact dependent on the individual state and the environment variables, $\boldsymbol{\chi}(a)$ is the individual state at age a and $\mathcal{F}(a)$ represents the probability that the individual survives until age a, which is defined as:

$$\mathcal{F}(a) \;=\; \exp\left(-\int_0^a \mu(\boldsymbol{\chi}(a), E)\, da\right)$$

with $\mu(\boldsymbol{\chi}(a), E)$ the individual's instantaneous mortality rate. The population-level impact on the environment now simply equals the product of the total population birth rate in equilibrium $\tilde{b}$ and the individual-level impact:

$$\tilde{I} = \tilde{b}\,\Gamma$$

The elements of `I[][]` and `impact[][]` therefore correspond one-to-one, such that for example in the PNAS model, in which `impact[0][0]` is defined as the individual feeding rate on the resource, the quantity `I[0][0]` equals the grazing rate of the entire population on the resource.

In the PNAS model, the first environment variable represents the resource density, which follows semi-chemostat growth in the absence of consumers. As a consequence, the resource R does NOT have an equilibrium value $\tilde{R} = 0$. The type of the first environment variable is therefore set to `GENERALODE` and its equilibrium condition is specified in line 34 of the code box above as `RHO*(RMAX - R) - I[0][0]`. The first term in this line of `C`-code implements the semi-chemostat dynamics in the absence of consumers and the quantity `I[0][0]` represents the grazing rate by the total population of consumers as mentioned above.

The second environment variable represents the predator density, the dynamics of which is described by the ODE

$$\frac{dP}{dt} \;=\; \left(\epsilon \frac{aB}{1 + T_h B} \;-\; \delta\right) P$$

Because $P = 0$ is a regular fixed point of this ODE, the type of the second environment variable is set to `PERCAPITARATE` on line 29 of the code box above, while its per capita growth rate $\epsilon a B/(1 + T_h B) - \delta$

is used to define its equilibrium condition on line 35. Notice in this respect that the population feedback quantity `I[0][1]` represents B, the total biomass of small juvenile consumers that are vulnerable to predation, which is calculated from the individual-level impact quantity `impact[0][1]` defined on lines 27-31 in code box 4.3.2.11.

Finally, the third environment variable in the PNAS model is the total biomass of small juvenile consumers B, which exerts a direct density-dependent effect on the mortality rate of small juvenile consumer individuals themselves, as it influences the predator functional response. The type of the third environment variable is therefore set to `POPULATIONINTEGRAL` on line 29 of the code box above. On line 36 of the code box above the equilibrium condition of this third environment variable is specified by identifying it with the population feedback quantity `I[0][1]`, which the program computes on the basis of the individual-level impact quantity `impact[0][1]` that represents the biomass of an individual consumer that is in the first life stage, where it is vulnerable to predation (lines 27-31 in the code box above).

> In case of multiple states-at-birth, individuals with different states-at-birth may have different impacts on their environment. The expected impact of an individual during its entire life on its environment is then given by an integral of the form:

$$\Gamma_j \;=\; \int_0^\infty \gamma(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)\,\mathcal{F}_j(a)\,da$$

in which $\gamma(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)$ quantifies the impact of an individual that is born with state $\boldsymbol{\phi}_j$ state dependent on its individual state at age a, $\boldsymbol{\chi}(a,\boldsymbol{\phi}_j)$, its state-at-birth $\boldsymbol{\phi}_j$ and the environment variables. $\mathcal{F}_j(a)$ now represents the probability that an individual born with state-at-birth $\boldsymbol{\phi}_j$ survives until age a, which is defined as:

$$\mathcal{F}_j(a) \;=\; \exp\left(-\int_0^a \mu(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)\,da\right)$$

with $\mu(\boldsymbol{\chi}(a,\boldsymbol{\phi}_j),\boldsymbol{\phi}_j,E)$ the individual's instantaneous mortality rate. $>>$ If the possible states-at-birth are given by the set $\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_m\}$, the individual-level impact is an m-dimensional vector $\Gamma = (\Gamma_1,\ldots,\Gamma_m)$. The population-level impact on the environment in this case equals the dot product of this vector Γ with the m-dimensional vector $\tilde{b}$, representing the equilibrium distribution of produced offspring over the possible states-at-birth $\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_m\}$ (refer to Diekmann, Gyllenberg, and Metz (2003) for details). $>>$ The program automatically computes the equilibrium distribution of produced offspring over the possible states-at-birth and uses it to compute the population-level impacts contained in `I[][]` from the individual-level impacts that have been specified in `impact[][]`.

4.4 Model analysis

4.4.1 Computation of curves and detections of bifurcation points

The software package allows to carry out 6 different types of computations of equilibria, corresponding to 6 different types of curves. These different types of computations are uniquely labeled with a 2- or 3-letter abbreviation code.

- `"EQ"`: This is the basic type of computation and hence the one that one usually starts out with. In this computational mode the software calculates the equilibrium of the PSPM as a function of a particular parameter over a range of values of that parameter. This type of computation hence

yields a type of curve that I will refer to as *equilibrium curve* (as opposed to the more general *bifurcation curve*).

During the computation of an equilibrium curve the software will detect 4 types of special points or bifurcation points:

- **Branching point**: A branching or transcritical bifurcation point is a point where two different equilibrium curves intersect. Generically, such a bifurcation occurs in a PSPM at a parameter value that represents the invasion or extinction threshold of a structured population and the two equilibrium curves are characterised by a zero (trivial) and non-zero (non-trivial) equilibrium value for a particular structured population, respectively. The program will report the occurrence of a branching point as `"BP #N"`, where `N` is the number of the structured population, for which the switch from a zero to non-zero equilibrium value occurs.
- **Environment branching point**: A branching or transcritical bifurcation point can also occur for an environment variable. The two equilibrium curves are then characterised by a zero (trivial) and non-zero (non-trivial) equilibrium value for a particular environment variable as opposed to a structured population. From the point of view of bifurcation theory, branching points that involve a zero and non-zero value of a structured population are the same as branching points that involve a zero and non-zero environment variable. The two types are only distinguished by the software, because they are computed differently. The program will report the occurrence of a branching point as `"BPE #N"`, where `N` is the number of the environment variable, for which the switch from a zero to non-zero equilibrium value occurs. Notice that environment branching points can only be detected for environment variables that are of the type `PERCAPITARATE` (see section 4.3.2.12 above).
- **Limit point**: At a limit point a saddle-node bifurcation occurs, in which 2 different equilibria in the model, an unstable saddle and a stable node, disappear at a particular threshold value of a parameter. At this parameter value the equilibrium curve reaches an extremum in the parameter values and hence bends back on itself. The program will report the occurrence of a limit point as `"LP"`.
- **Evolutionary fixed point**: The equilibrium condition for a structured population model can be expressed as:

  $$R_0(p, E(p)) \;=\; 1$$

  in which p is one of the model parameters. The quantity $R_0(p, E(p))$ refers to the expected number of offspring that an individual of the structured population will produce during its lifetime, in an environment that is characterized by the environment variables $E(p)$. The parameter p may directly influence the value of $R_0(p, E(p))$ but also indirectly, because it may have an impact on the equilibrium values of the environment variables. The software uses the condition above to compute the equilibrium of a PSPM (see also chapter 10).

  In an evolutionary setting, in which mutations in the parameter p can occur and selection acts to increase or decrease the parameter p over evolutionary time, higher values of p are selected for if $\partial R_0/\partial p > 0$, while lower values of p are selected for if $\partial R_0/\partial p$ is negative. Any parameter value where $\partial R_0/\partial p = 0$ is a fixed point of the evolutionary process, as for this parameter value the selection gradient for the parameter p is 0 (Metz et al. 1996, Geritz et al. (1998), Diekmann, Gyllenberg, and Metz (2003)). The software will detect these evolutionary fixed points. In addition, the software will compute the second-order partial derivatives of R_0 to classify the evolutionary fixed point as a convergent stable strategy

(CSS), an evolutionary repellor (ERP) or an evolutionary branching point (EBP) (Geritz et al. 1998).

It should be noted that evolutionary fixed points are normal equilibrium points of the dynamics system, as opposed to special bifurcation points, since the (ecological) dynamics of the model do not change at the critical parameter value. However, from en evolutionary perspective these points are special. They moreover play a key role in the theory of Adaptive Dynamics (Metz et al. 1996, Geritz et al. (1998), Diekmann, Gyllenberg, and Metz (2003)). It is for this reason that the software reports them as special points.

The remaining 5 types of computations that the software can carry out all involve one of the special, bifurcation points that the software detects during the computation of an equilibrium curve. During the computation of the following curves always 2 model parameters are varied, hence these computations are referred to as *two-parameter bifurcations*, as opposed to the *one-parameter bifurcation* of an equilibrium curve.

- `"BP"`: In this computational mode the software computes the location of a branching point of a structured population as a function of 2 model parameters. The resulting line can hence be interpreted as the invasion or extinction boundary of the structured population. This computation should start from an initial point close to a (detected) branching point.

- `"BPE"`: Similarly, in this computational mode the software computes the location of an environment branching point as a function of 2 model parameters. The resulting line can hence be interpreted as the boundary separating a parameter region with a zero equilibrium value for the environment variable from a region with a non-zero equilibrium value of the environment value. This computation should start from an initial point close to a (detected) environment branching point.

- `"LP"`: In this computational mode the software computes the location of a limit point as a function of 2 model parameters. The resulting line can hence be interpreted as the boundary separating a parameter region with (at least) two equilibrium states from a parameter region where these two specific equilibrium states do not occur. This computation should start from an initial point close to a (detected) limit point.

- `"ESS"`: The value of a particular parameter for which an evolutionary fixed point occurs is generically referred to as an ESS parameter value. In this computational mode the software computes the location of such an ESS parameter value as a function of the bifurcation parameter. This computational mode can hence be used to investigate how the evolutionary optimal value of one model parameter depends on the value of a second parameter. Curves of this type correspond to the evolutionary isocline of the ESS parameter as a function of the bifurcation parameter. `"ESS"` computations are, however, not limited to a single parameter having its ESS value, the program can also compute curves of equilibria, in which multiple model parameters are at their ESS value. The bifurcation parameter, which parameterises the curve, is not one of these ESS parameters. This computation should start from an initial point close to a (detected) evolutionary fixed point.

- `"PIP"`: This computation also starts from an initial point close to a (detected) evolutionary fixed point. In the theory of adaptive dynamics that focuses on evolutionary analysis the pairwise invasibility plot or PIP plays an important role (Dieckmann 1997, Metz et al. (1996)). The PIP is a two-dimensional plot, spanned by the parameter value of a resident type on the x-axis and the parameter value of a mutant type on the y-axis. The plot indicates for which parameter values the mutant type has a positive (negative) growth rate and hence can (can not) invade in the equilibrium as set by the resident type. In the computational mode `"PIP"` the software

computes the boundary between the parameter regions, for which the mutant has a positive and a negative growth rate.

4.4.2 Arguments of the PSPMequi function

Once the model has been implemented, you can proceed carrying out its analysis with the R-function PSPMequi. The syntax for calling PSPMequi is shown in the R command box below, which lists all possible arguments to the function as well as their default values.

```
> output <- PSPMequi(modelname = NULL, biftype = NULL, startpoint = NULL, stepsize = NULL,
                     parbnds = NULL, parameters = NULL, options = NULL, clean = FALSE,
                     force  = FALSE, debug  = FALSE)
```

The first 5 arguments of the function PSPMequi are needed for the software to function properly and are hence obligatory. The last 5 arguments are optional. The last 3 arguments are boolean arguments which have to be defined as either TRUE or FALSE, for example by passing as argument to the function PSPMequi the argument clean = TRUE.

The first 7 arguments to the PSPMequi function are the following:

1. The first argument to the R-function PSPMequi is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension '.R' or '.h' as part of the file name, the PSPMequi function will automatically try to locate the appropriate file, checking first for a file implemented in C (with an extension '.h') and subsequently for a file implemented in R (with an extension '.R'). If both a file with an extension '.h' and a file with an extension '.R' are found, the program will use the first one. The program can be forced to use the file with an extension '.R' by including the extension explicitly as part of the filename. The R-command to analyze the PNAS model that will be used as illustrations below will therefore all take "PNAS2002" as their first argument. If the file specifying the PSPM can not be found in the current directory, the PSPMequi function will ask the user to search in the package directory for a model file with the specified name.

2. The second argument to the PSPMequi function determines which type of computation should be carried out for the particular model. These types of computation are discussed in section 4.4.1 above. This string argument should hence be either "BP", "BPE", "EQ", "LP", "ESS" or "PIP".

3. The third argument is the initial point of the computation. This initial point should be close to a solution point for the selected computation, that is, close to an equilibrium point, a branching point, an environment branching or a limit point for an "EQ", "BP", "BPE" and a "LP" computation, respectively. For either a "ESS" or a "PIP" computation the initial point should be close to an evolutionary fixed point.

   The initial point should be a (row) vector with the proper dimension. For equilibrium computations (type "EQ") this vector in general consists of the initial value of the model parameter to vary, the estimated equilibrium values for all the environment variables and the estimated values of the birth rate for all the structured populations in the model, in the following order:

   c(<*parameter*>,<*environment variables*>,<*population birth rates*>)

   However, environment variables that have been explicitly specified with the program option "envZE" as having a zero equilibrium value and birth rates of populations that have been explicitly specified with the program option "popZE" to be in a zero equilibrium state (see the description of these options under point 7 below), should be omitted from this vector of initial values.

   For all two-parameter types of bifurcation computations ("BP", "BPE", "LP", "ESS" and "PIP") the value of the second model parameter should be appended as last element to the vector of initial values, which therefore has the format:

`c(`<*parameter*>,<*environment variables*>,<*population birth rates*>,<*parameter 2*>`)`

For `"ESS"` computations with multiple parameters having their ESS value this vector has to be extended with the initial estimate of the ESS value for each of these model parameters.

Also for these computations holds that environment variables that have been explicitly specified with the program option `"envZE"` as having a zero equilibrium value and birth rates of populations that have been explicitly specified with the program option `"popZE"` to be in a zero equilibrium state (see the description of these options under point 7 below), should be omitted from this vector of initial values. In addition, in case of a continuation of a branching point, either for a structured population model (type `"BP"`) or for an environment variable (type `"BPE"`), the zero value for the birth rate of the structured population or the environment variable (specified with the program options `"popBP"` and `"envBP"`, respectively; see below), for which the branching point occurs, should also be omitted from the vector of initial values.

4. The fourth argument to the `PSPMequi` function determines the step size along the computed curve. The absolute spacing between subsequent solution points computed on the curve is difficult to predict, as it is determined by both the step size and how quickly the different variables change along the curve. The step size can be either positive or negative, while step sizes of smaller absolute value will lead to the computation of solution points that are more closely spaced together.

5. The fifth argument to the `PSPMequi` function determines which of the model parameters should be varied during the computation and at which parameter values the computations should stop. This information should be specified by a (row) vector, which for every parameter to be varied specifies a triplet of values including the index of the parameter, its minimum and its maximum value at which the computation should stop. For equilibrium computations (type `"EQ"`) the vector should hence have the following format:

   `c(`<*index 1*>,<*minimum 1*>,<*maximum 1*>`)`

   The first element of the vector indicates the index of the parameter in the vector `DefaultParameters` (in `R`, see section 4.3.1.4) or in the array `parameter` (in `C`, see section 4.3.2.2) to vary, while the final two elements of the array indicate the minimum and maximum value of the parameter. The computation of the equilibrium curve as a function of the model parameter stops, whenever the minimum or maximum parameter value is reached.

   For all two-parameter type of computations (`"BP"`, `"BPE"`, `"LP"`, `"ESS"` and `"PIP"`) the vector should be extended with the index of the second parameter to vary in the computation as well as the minimum and the maximum value of this parameter, at which to stop the curve computation. The vector with parameter information has therefore in this case the format:

   `c(`<*index 1*>,<*minimum 1*>,<*maximum 1*>,<*index 2*>,<*minimum 2*>,<*maximum 2*>`)`

   For `"ESS"` computations with multiple parameters having their ESS value this vector has to be extended with a triplet of values for each of these model parameters with the triplet specifying the index of the particular ESS parameter as well as its minimum and the maximum value. The number of triplets should correspond with the number of initial estimates for the ESS values of parameters as specified in the third argument to the function.

6. The sixth, optional argument of the `PSPMequi` function is a (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (the length of the vector `DefaultParameters` in `R`, see section 4.3.1.4, or the value of `PARAMETER_NR` in `C`, see section 4.3.2.1). When of this length the values will replace the default values of the parameters that are listed in the model specification file (see code block 4.3.1.4 or

4.3.2.2 for an example). If the array used for this sixth argument is not of the correct length or when it is not specified at all, it will simply be ignored.

7. The seventh, optional argument of the `PSPMequi` function is a (row) vector of string elements, containing possible options that modify the behavior of the computational module. Some of the options require a value and hence occur as a pair of option name and option value, while others occur on their own. Options can be specified in any order, but the option value should always immediately follow after the option name. All option values refer to indices of either environment variables, structured populations or individual state variables. Notice, that this index value follows the `C`-convention of ordering arrays starting at 0 (as opposed to `R` where array indices start at 1). Multiple options can be included into the vector like:

   `c("`*name 1*`", "`*value 1*`","`*name 2*`","`*value 2*`","`*name 3*`","`*value 3*`")`

   Possible options are:

   i) Option pair `c("envBP", "i")`: This option pair is only relevant for continuations of a branching points or transcritical bifurcation in an environment variable (curve type `"BPE"`). The option value `"i"` determines the index of the environment variables, of which to continue the transcritical bifurcation as a function of 2 parameters. Notice that this computation can only be carried out for environment variables that are of the type `PERCAPITARATE` (see section 4.3.1.3 or section 4.3.2.12 above).

   ii) Option pair `c("popBP", "i")`: This option pair is only relevant for continuations of branching points or transcritical bifurcations of a structured population (curve type `"BP"`). The option value `"i"` determines the index of the population, of which to continue the transcritical bifurcation as a function of 2 parameters.

   iii) Option pair `c("popEVO", "i")`: This option pair is relevant for the computation of the selection gradient during equilibrium continuations (curve type `"EQ"`), continuations of evolutionary fixed points (curve type `"ESS"`) and the construction of pairwise invasibility plots (curve type `"PIP"`). The option value `"i"` determines the index of the population, for which to compute the selection gradient or in which the evolutionary singularity occurs that should be computed as a function of 2 parameters.

   iv) Option pair `c("parEVO", "i")`: This option pair is only relevant for the computation of the selection gradient during equilibrium continuations (curve type `"EQ"`) and then only when the option pair `c("popEVO", "i")` is also specified. In this case the index `"i"` determines the index of the parameter for which to compute the selection gradient. The default is to compute the selection gradient with respect to the bifurcation parameter. Specifying this index does compute the selection gradient for a particular parameter, but detection of evolutionary singular points is only possible if the bifurcation parameter is the evolutionary parameter.

   v) Option pair `c("envZE", "i")`: This option pair can be specified several times as part of the option vector of strings. Including this option instructs the computational module to set the value of the environment variable with index `"i"` equal to 0 during the computations of the fixed point problem that determines the curve. In addition, the equilibrium condition for this environment variable (as, for example, specified in `R` code block 4.3.1.9 or `C` code block 4.3.2.12) is ignored and hence not included as condition to hold in the particular equilibrium point. Notice that this can only occur for environment variables that are of the type `PERCAPITARATE` or `POPULATIONINTEGRAL` (see section 4.3.1.3 or section 4.3.2.12 above). Forcing an environment variable to have a zero equilibrium value as opposed to specifying a value of 0 for it as part of the initial point of the computation, allows for the proper

detection and handling of branching or transcritical bifurcation points in this environment variable. Omitting this option for an environment variable, but providing instead a value of 0 as part of the initial point of the computation, may lead to the proper computation of an equilibrium curve, in which the environment variable has a 0 value, but may also lead to numerous, spurious messages about branching points in this variable.

vi) Option pair `c("popZE", "i")`: This option pair can be specified several times as part of the option vector. Including this option forces the computational module to assume that the structured population with index `"i"` in the model is in a zero equilibrium state for the curve that has to be computed. This is the only way to compute an equilibrium curve with a zero equilibrium state for a particular parameter. Even if a value of 0 would be specified for the birth rate of a population as part of the initial point of the computation, the software would compute the equilibrium curve with a non-zero (non-trivial) equilibrium state for this population. Notice that if a structured population is forced to be in a zero equilibrium state by using the `"popZE"` option, a zero equilibrium state should also be enforced for all the environment variables that represent integrals over this population distribution (that are hence of the type `POPULATIONINTEGRAL`).

vii) Option pair `c("isort", "i")`: This option modifies the output of the equilibrium state of the populations that are stored in an output file with a name of the form `<Modelname>-<Type>-<NNNN>.csb` (see below). By default the computational module reports the information about the stable population state distributions by subdividing the axis of the first state variable (the one with index `"0"`) in 100 subintervals of equal length and reporting the statistics for the cohort of individuals within each subinterval. By using the option `"isort"` the default choice to use the first individual state variable for this subdivision can be changed to the second, third, and so on. Therefore, passing `c("isort", "0")` as option vector to the `PSPMequi` function is the same as the default behaviour: the first individual state variable is used for the subdivision and ordering of the population state distribution, while passing `c("isort", "1")` would use the second individual state variable for this purpose. Also notice that the number of subdivisions of the individual state variable can be redefined by assigning the dimension `COHORT_NR` a value different from 100 (see code block 4.3.1.2 or 4.3.2.1 as well as section 9.3).

viii) Option `c("noBP")`: Specifying the option `"noBP"` instructs the program not to test for the occurrence of branching points (`"BP"`), when computing an ecological equilibrium as a function of a single model parameter. The advantage of specifying this option is that execution speed increases, as testing for a branching point is computationally demanding. The disadvantage is that the exact location of branching points will not be reported.

ix) Option `c("noLP")`: Specifying the option `"noLP"` instructs the program not to test for the occurrence of limit points (`"LP"`), when computing an ecological equilibrium as a function of a single model parameter. The advantage of specifying this option is that execution speed increases, as testing for a limit point is computationally demanding. The disadvantage is that the exact location of limit points will not be reported.

x) Option `c("single")`: When the option `"single"` is specified as part of the option cell array, the program will only compute a single point of the equilibrium curve, for the initial value of the bifurcation parameter. The program will not continue the equilibrium curve originating from this first point. This option is hence useful for when only a single equilibrium is of interest.

xi) Option `c("test")`: The last possible option that can be passed to the `PSPMequi` function as part of the option vector is the `"test"` option. This invokes the computational module in testing mode, which implies that only a single integration of the individual life history is

carried out and no iteration to locate a fixed point of a set of equations is performed. In testing mode the computational module reports on the dynamics of the individual state variables, the survival, the cumulative impact on the environment and the expected number of offspring produced by an individual during its different life stage as well as over its entire life. Testing mode is very useful to discover whether or not the model implementation gives sensible results or not.

Three other optional arguments can be passed to the `PSPMequi` function: `clean`, `force` and `debug`. These are all boolean arguments that hence have to be passed to the `PSPMequi` function as *<option name>*`=TRUE` or *<option name>*`=FALSE`, the latter being the default value of all options (Specifying these options as argument is hence only useful when setting them equal to `TRUE`). Unlike the previous arguments, which all modify the computations to be performed, these options modify the behavior of the `PSPMequi` function itself, in particular the compilation of the model specific file into a dynamic library module that can be executed from `R`. Also unlike all the previous arguments that can be passed, these arguments can be passed in any order and at any position, the `PSPMequi` function will filter these 3 optional arguments from the argument list before passing the filtered argument list to the computational routine.

- Option `clean`: When `clean=TRUE` is passed as argument, this argument instructs the `PSPMequi` function to delete all result files that have been generated during previous calculations with the model. These result files have names of the form `<Modelname>-<Type>-<NNNN>.<ext>`, in which `<Modelname>` refers to the name of the model (i.e. `PNAS2002` in the example model presented in previous sections), `<Type>` refers to the type of continuation that has been performed, i.e. `BP`, `BPE`, `EQ`, `ESS`, `LP` or `PIP`, `<NNNN>` is a unique number that distinguishes consecutive computations of the same type of curve with the same model, and `<ext>` is one of the extensions `.bif`, `.csb`, `.err` or `.out`. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function `PSPMclean()` to delete all `.bif`, `.err`, `.csb` and `.out` files and/or all executable files that are present in the current working directory.

- Option `force`: When `force=TRUE` is passed as argument, it instructs the `PSPMequi` function to force re-compilation of the model specific file into a dynamic library module that can be executed by `R`. This option will usually not be needed by normal users, as the `PSPMequi` function automatically recompiles the computational module when the model specific file with an `'.h'` extension is more recently changed than the compiled dynamic library file. However, if for some unclear reason this automatic recompilation fails, the `force` option can be used to initiate re-compilation.

- Option `debug`: When `debug=TRUE` is passed as argument, it instructs the `PSPMequi` function to turn on debugging flags while compiling the model specific file into a dynamic library module. This option can be useful to detect programming mistakes in the model-specific file that are otherwise hard to track down. The downside is that depending on the version of `R` that is used, turning on debugging flags during compilation may generate a lot of output, including warnings about standard files of the operating system that are perfectly correct. It is hence not so easy to spot among all these messages the warnings that relate to the model-specific code that has been implemented.

4.4.3 Output variables of the `PSPMequi` function

When calling the `PSPMequi` function it first compiles the model-specific file called `<Model>.h` using the `R` command `R CMD SHLIB` into a dynamically loadable library file, which can subsequently be executed.

When the model is implemented in `R` instead of in `C`, the `PSPMequi` function compiles all necessary numerical routines in the package with the appropriate dimension settings that are taken for the `<Model>.R` model file. These compilation steps are only carried out when the dynamically loadable library file (called `<Model>equi.so` on my system on Mac OS X and Linux systems) does not exist, or when the model-specific file has been changed since the last compilation of the executable. Furthermore, the compilation step is forced by the invocation of `PSPMequi` with the additional argument `force=TRUE` as discussed in the previous section. For example, for the PNAS model, the `C` implementation of which is specified in the file `PNAS2002.h`, the dynamically loadable library file is called `PNAS2002equi.so` (on other operating systems this file may be called `PNAS2002equi.dll`). Following successful compilation the `PSPMequi` function executes the compiled, computational module with the arguments passed to it.

The computational module generates on execution a single list object as output with up to 4 member elements (see the help page on `PSPMequi` using `?PSPMequi`). The first element of the output list, `output$curvepoints`, contains the numerical information of the points along the computed curve (assuming that the list is assigned to a variable called `output`). This variable `output$curvepoints` is a matrix, in which each row represents one solution point along the curve. The columns contain the value of the parameter(s) that have been varied, the equilibrium value of all environment variables, the equilibrium value for the birth rate of all structured populations in the problem, the equilibrium value of all interaction variables defined in the routine `Impact()` (see R code block 4.3.1.7 or C code block 4.3.2.11), the per capita growth rate of all environment variables for which this is relevant (those of the type `PERCAPITARATE`, see section 4.3.1.3 or section 4.3.2.12), for each of the structured populations the expected number of offspring produced by an individual during its lifetime (R_0), for the structured population with index `popEVO` (if this index is defined via the option vector) the derivative of its R_0 value with respect to the evolutionary parameter (determined by the option `parEVO`, by default the bifurcation parameter) and finally the norm of the right-hand side of the system of equations that is solved. The latter quantity (referred to as RHS norm) measures how close the computed solution point is to the true solution. The derivative of the R_0 value for a structured population with respect to the bifurcation parameter can be used as an indicator for evolutionary change: positive and negative values of this derivative indicate that there is selection for larger and smaller values of the bifurcation parameter, respectively.

The column layout just described pertains to computations of equilibrium curves. For other types of computed curves the number of columns in the output variable `output$curvepoints` is different. For all curves that depend on two parameters (curve types `"BP"`, `"BPE"`, `"LP"` and `"PIP"`) the value of the second parameter is inserted as an additional column after all equilibrium values for the birth rates of the structured populations. For `"ESS"` curves additional columns are inserted after all these equilibrium birth rate values for each of the parameters that is forced to its evolutionary stationary value along the curve. In addition, preceding the final column with the RHS norm additional columns are added for the second-order partial derivatives of R_0 of the structured population with index `popEVO` with respect to the resident and mutant value of each of the evolutionary parameters, respectively (see sections 6.1 to 6.3 for more details). These partial derivatives characterize the evolutionary fixed point as a convergent stable strategy (CSS), an evolutionary repellor (ERP) or an evolutionary branching point (EBP).

When the `PSPMequi` function finishes, it prints textual information about the computation that has been carried out. This text also contains a header line indicating which column of the output contains which particular value (see the section 4.4.4 below).

The second member element of the output list, `output$curvedesc`, (see the help page on `PSPMequi` using `?PSPMequi`), which is always produced by the computational module irrespective of the type of curve computation that is carried out, contains the description of the executed calculation, which includes the command-line that is used for the invocation of the computational routine, the values of all parameters used for the current computation and a header line indicating the meaning of all

the output variables produced by the computational module. This textual information is also printed to the R console at the end of calculations. In fact, the PSPMequi function prints its report on the calculations by execution of the statement cat(output$curvedesc, sep='\n'). More details about the content of the description variable output$curvedesc is provided in the section below discussing the example of a model analysis using the PSPMequi function.

The third and fourth member elements of the output list are only non-empty for computations of equilibrium curves (type "EQ"). The third output variable output$bifpoints (see the help page on PSPMequi using ?PSPMequi) contains the same type of information as the first output variable output$curvepoints, but now only for the detected bifurcation points along the computed curve. Finally, the fourth and last output variable output$biftypes (see the help page on PSPMequi using ?PSPMequi) is a vector with string values that indicate the type of bifurcation point detected. These strings can be, for example, "BP #0", "BPE #0", "LP" or "CSS #0". Each element in the vector output$biftypes characterizes the corresponding row in the output variable output$bifpoints.

4.4.4 An example session using the PSPMequi function

To illustrate the use of the PSPMequi function I will discuss the analysis of the PNAS model, presented in section 4.2. The statements below are taken from one of the demos, called "deRoosPersson", that is included with this R package. This demo executes in addition to the statements represented here, R commands to visualize the computed results in graphs. It is therefore recommended to run the demo, using the command demo("deRoosPersson", echo=FALSE) at the same time as reading the explanation in this section (but notice first the package has to be loaded via library("PSPManalysis")). The demo uses the C implementation of the PNAS model (see section 4.3.2).

Starting the analysis of a PSPM from some random initial values to search for the equilibrium values of environment variables and population birth rates is often not a very successful strategy. Most likely, the initial point will be too far off a solution point, which might cause the software not to converge to the solution. A better alternative is to start with a trivial equilibrium of the model, the value of which is known on beforehand. For example, it is biologically realistic to assume that at very high mortality or at very low food conditions a population is extinct. Such an extinct state is often easily characterized and hence provides a useful starting point.

The analysis of the PNAS model is therefore started from the trivial equilibrium, which is stable for very low values of the resource productivity R_{max}. In this equilibrium only the resource density has a non-zero value equal to its maximum $\tilde{R} = R_{max}$, while both the consumer and the predator equilibrium are in a zero equilibrium state. For this reason, the PSPMequi function is invoked with the option vector c("popZE", "0", "envZE", "1", "envZE", "2"), which enforces this zero equilibrium state for the structured population with index 0 (the consumer), the environment variable with index 1 (the predator) and the environment variable with index 2 (the biomass of small consumers that are vulnerable to predation). The latter is obviously in a zero equilibrium state, because the variable represents an integral over the population distribution of the consumer. More generally, if a structured population is assumed to be in a zero equilibrium state by using the "popZE" option, a zero equilibrium state should also be enforced for all the environment variables that represent integrals over this population distribution (that are hence of the type POPULATIONINTEGRAL). Because consumer and predator are assumed to be in a zero equilibrium state the initial point for the call to the PSPMequi function shown in the box 4.4.4.A below consists of only 2 elements, R_{max} and the initial guess for $\tilde{R}$, which are both taken equal to $1.0 \cdot 10^{-6}$. The parameter with index 1, which corresponds to R_{max}, is the one to be varied in positive direction with step size 0.5 over the range 0 to $4.0 \cdot 10^{-4}$.

> Notice that if the R implementation of the PNAS model would have been used for this demonstration the index of the parameter to vary should have been specified as 2 instead of 1, since vector indices in R start at 1 whereas in C they start at 0. As an alternative

to specifying the index of the parameter, if the model is implemented in `R`, the name of the parameter in the parameter vector `DefaultParameters` (see code block 4.3.1.4) could have been specified as a string `"Rmax"` as first element of the fifth argument in the call to the `PSPMequi` function shown below.

The sixth argument in the call to the `PSPMequi` function shown below is left undefined by specifying it as `NULL`. This argument represents the parameter values to be used for the computation. Because a vector is not specified, the argument is ignored and the default parameter values, as specified in code block 4.3.2.2, are used. This sixth argument will also be left undefined in all further calls to the `PSPMequi` function discussed below and will therefore from hereon be ignored.

Command box 4.4.4.A

```
> output1 <- PSPMequi("PNAS2002", "EQ", c(1.0E-06, 1.0E-06), 0.5, c(1, 0, 4E-4), NULL,
+                     c("popZE", "0", "envZE", "1", "envZE", "2"), clean=TRUE, force=TRUE);

Building executable PNAS2002equi.so ...

<...compilation output lines suppressed in this box...>

  1.00000000E-06  1.00000000E-06
  1.35355339E-06  1.35355339E-06
  1.70710678E-06  1.70710678E-06
<...output lines suppressed in this box...>
  8.77817459E-06  8.77817459E-06
  8.85690312E-06  8.85690312E-06  ****  BP #0   ****
  9.13172798E-06  9.13172798E-06
<...output lines suppressed in this box...>
  3.34047294E-04  3.34047294E-04
  3.69402633E-04  3.69402633E-04
  4.04757972E-04  4.04757972E-04


#
# Executing : PSPMequi("PNAS2002", "EQ", c(1E-06, 1E-06), 0.5, c(1, 0, 0.0004), NULL, c("popZE", "0", "envZE",
    "1", "envZE", "2"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  1E-06          Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
#
#           1:Rmax       2:E[0] 3:E[1] 4:E[2] 5:b[0] 6:I[0][0] .. 10:pcgE[1]     11:R0[0]  12:RHS norm
#
> output1$curvepoints
                V1           V2     V3     V4     V5        V6 ..        V10          V11           V12
 [1,] 1.000000e-06 1.000000e-06      0      0      0         0 ..      -0.01    0.0000000 0.000000e+00
 [2,] 1.353553e-06 1.353553e-06      0      0      0         0 ..      -0.01    0.0000000 4.235165e-17
 [3,] 1.707107e-06 1.707107e-06      0      0      0         0 ..      -0.01    0.0000000 4.235165e-17
<...output lines suppressed in this box...>
[70,] 2.986920e-04 2.986920e-04      0      0      0         0 ..      -0.01 4355.7372537 1.084202e-16
[71,] 3.340473e-04 3.340473e-04      0      0      0         0 ..      -0.01 4430.1631161 5.421011e-17
[72,] 3.694026e-04 3.694026e-04      0      0      0         0 ..      -0.01 4491.4630291 8.131516e-16
[73,] 4.047580e-04 4.047580e-04      0      0      0         0 ..      -0.01 4542.8245742 1.084202e-16
> output1$bifpoints
                V1           V2     V3     V4     V5        V6 ..        V10          V11           V12
 [1,] 8.856903e-06 8.856903e-06      0      0      0         0 ..      -0.01            1             0
> output1$biftypes
```

```
[1] "BP #0"
```

In the output shown in the box above a large number of intermediate output lines and several intermediate columns have been deleted from the output. Consult the listing in your R console for the complete output of the commands executed.

After the specific call shown, the PSPMequi function first cleans all previous output file (notice the option clean=TRUE) and subsequently compiles the computational module using the R command R CMD SHLIB into a dynamically loadable library file (notice the somewhat superfluous option force=TRUE). Subsequently, the compiled module is executed with the obligatory arguments that the PSPMequi function passes on. The computational module computes the particular equilibrium curve over the required range of the parameter with index 1 (representing R_{max}; see code block 4.3.2.2). At the end of the computation the PSPMequi function prints a textual summary of the computation that has been executed. This is summary is in fact the content of the second element of the output list of the function, output1$curvedesc, which is printed to the R console using the statement cat(output1$curvedesc, sep='\n'). Apart from printing the exact command-line that has been used to start the computation, the values of the parameters are printed using the meaningful, model-specific names, as set in the code block 4.3.2.2. Furthermore, a header line is printed with a short description of each of the columns in the output matrix output1$curvepoints. As shown in the box above, executing the command output1$curvepoints shows the value of the various columns in this output matrix.

Most importantly, halfway during the computation of the trivial equilibrium the software reports that a branching point has been located, indicated with "BP #0". This branching or transcritical bifurcation point corresponds to the invasion threshold of the consumer. The exact data about the branching point are returned as the third member element of the output list, output1$bifpoints. Displaying output1$bifpoints reveals that it has the same layout as the output matrix output1$curvepoints, but only contains a single row with data for the branching point. Notice that the value of R_0, the expected number of offspring produced by an individual consumer during its lifetime, is exactly equal to 1 in this branching point (as it should be), whereas it is smaller and larger than 1 for lower and higher values of R_{max}, respectively. The corresponding element of the fourth output variable, assigned to the member element output1$biftypes of the output list, contains the descriptive string "BP #0" that is also printed to the console on detection of the branching point.

In addition to executing the call to the PSPMequi function shown in box 4.4.4.A, the demo script called "deRoosPersson" also uses the output variables output1$curvepoints, output1$bifpoints and output1$biftypes to generate a plot of the computed results.

The next step in the analysis of the PNAS model starts from the detected transcritical bifurcation point that is stored in output1$bifpoints. Starting from that point the call to PSPMequi shown in the next R command box computes the equilibrium curve with a non-zero equilibrium state of the consumer, while the predator is still assumed to have a zero equilibrium value. Because the absence of the predator ensures that the third environment variable, representing the total biomass of small consumers that the predator forages on, does not influence the equilibrium state, this third environment variable is also ignored. The PSPMequi function is therefore called with the option array c("envZE","1","envZE","2"). As before, the curve is computed as a function of the parameter with index 1, which corresponds to R_{max}, with step size 0.2 over the range of R_{max} values between 0 and to $4.0 \cdot 10^{-4}$. As initial point of the computation the R_{max}-value and the equilibrium values of the resource density and the population birth rate in the bifurcation point are used, which correspond to the first, second and fifth element in output1$bifpoints (although output1$bifpoints[5] is of course equal to 0).

Command box 4.4.4.B

```
1 > output2 <- PSPMequi("PNAS2002", "EQ", output1$bifpoints[c(1,2,5)],0.2,
      c(1,0,4E-4),NULL,c("envZE","1","envZE","2"))
```

```
Dynamic library file PNAS2002equi.so is up-to-date

  8.85690312E-06  8.85690312E-06  0.00000000E+00
  9.05689403E-06  8.85690312E-06  1.90665832E-09
  9.25688494E-06  8.85690312E-06  3.81331663E-09
<...output lines suppressed in this box...>
  2.51840533E-04  8.85690312E-06  2.31653906E-06
  2.53602314E-04  8.85690312E-06  2.33333540E-06  ****  BPE #1  ****
  2.66316121E-04  8.85690312E-06  2.45454534E-06
<...output lines suppressed in this box...>
  3.82120831E-04  8.85690312E-06  3.55859558E-06
  3.96596420E-04  8.85690312E-06  3.69660186E-06
  4.11072009E-04  8.85690312E-06  3.83460814E-06

#
# Executing :
    PSPMequi("PNAS2002","EQ",c(8.8569E-06,8.8569E-06,0),0.2,c(1,0,0.0004),NULL,c("envZE","1","envZE","2"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  8.8569E-06     Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
#
# 1:           Rmax       2:E[0]       3:E[1]       4:E[2]       5:b[0]       10:pcgE[1] 11:R0[0]  12:RHS norm
#
> output2$curvepoints
                  V1           V2           V3           V4           V5 ..           V10      V11           V12
 [1,]  8.856903e-06 8.856903e-06            0            0 0.000000e+00 .. -1.000000e-02        1 4.936400e-08
 [2,]  9.056894e-06 8.856903e-06            0            0 1.906658e-09 .. -9.991810e-03        1 4.936400e-08
 [3,]  9.256885e-06 8.856903e-06            0            0 3.813317e-09 .. -9.983620e-03        1 4.936401e-08
<...output lines suppressed in this box...>
[91,]  3.821208e-04 8.856903e-06            0            0 3.558596e-06 ..  5.235110e-03        1 5.005141e-08
[92,]  3.965964e-04 8.856903e-06            0            0 3.696602e-06 ..  5.824070e-03        1 5.010536e-08
[93,]  4.110720e-04 8.856903e-06            0            0 3.834608e-06 ..  6.412900e-03        1 5.016130e-08
> output2$bifpoints
                  V1           V2           V3           V4           V5 ..           V10      V11           V12
 [1,]  0.0002536023 8.856903e-06            0            0 2.333335e-06 .. -9.538660e-11        1 5.628726e-09
> output2$biftypes
[1] "BPE #1"
```

As in the previous `R` command box a number of output lines and columns have been suppressed to fit the page width of this manual. Please consult the listing in your `R` console for the complete output of the commands executed.

The layout of the output and the output variables of the call to `PSPMequi` shown in box 4.4.4.B is similar to the output as discussed following `R` command box 4.4.4.A. The data of the computation are stored in the output list `output2`. The data of the computed equilibrium points are stored in the element `output2$curvepoints`. Halfway during the computation of the equilibrium the software reports that a branching point has been located for the environment variable with index 1, indicated with `"BPE #1"`. This branching or transcritical bifurcation point corresponds to the invasion threshold of the predator. For R_{max}-values above this threshold the predator can invade the equilibrium of the consumer population, but it fails to invade for lower R_{max}-values. The exact data about this branching point are contained in `output2$bifpoints`. Displaying `output2$bifpoints` reveals that it has the same layout as the output matrix `output2$curvepoints`, but only contains a single row with data for the branching point. Notice that the population growth rate of the predator, shown in column 10 labeled `pcgE[1]` of the output, is equal to 0 at the detected branching point (as it should be), whereas

it is smaller and larger than 0 for lower and higher values of R_{max}, respectively. The last element of the output list, `output2$biftypes`, contains the descriptive string `"BPE #1"` that is also printed to the console on detection of the branching point.

As before, the demo script called `"deRoosPersson"` also uses `output2$curvepoints`, `output2$bifpoints` and `output2$biftypes` to plot the computed results of the call to the `PSPMequi` function shown in box 4.4.4.B as additional curves in the graphs that it had generated previously.

The final step in this part of the analysis of the PNAS model starts from the detected transcritical bifurcation point of environment variable 1, representing the predator population, which is stored in the output variable `output2$bifpoints`. Starting from that point the call to `PSPMequi` shown in the next `R` command box computes the equilibrium curve with a non-zero equilibrium state of the consumer and predator. All environment variables influence this equilibrium state, hence the `PSPMequi` function is called without specifying any options. As before, the curve is computed as a function of the parameter with index 1, which corresponds to R_{max}, with step size -0.1 over the range of R_{max} values between 0 and to $4.0 \cdot 10^{-4}$. The choice of a negative step size is arrived at by trial and error. Choosing a positive step size to start the continuation from this point would have quickly shown that the equilibrium predator density would turn negative, whereas this equilibrium density increases from 0 to positive values with a negative step size. The transcritical bifurcation in this invasion point of the predator is hence *subcritical*.

As initial point of the computation the data of the branching point contained in `output2$bifpoints` are used. This initial point should contain appropriate values for the bifurcation parameter R_{max}, the equilibrium values of the resource density, the predator density and the biomass density of small, vulnerable consumers as well as the population birth rate in the bifurcation point. Normally, appropriate starting values for these variables are to be found in the first 5 elements of the vector `output2$bifpoints`. Inspection of `output2$bifpoints[1:5]`, however, shows that both `output2$bifpoints[3]` and `output2$bifpoints[4]` are equal to 0:

```
> output2$bifpoints[1:5]
[1] 2.536023e-04 8.856903e-06 0.000000e+00 0.000000e+00 2.333335e-06
```

Obviously, `output2$bifpoints[3]` is equal to 0 as it represents the zero equilibrium value of the predator in the consumer-resource equilibrium curve that is computed with the call to `PSPMequi` shown in `R` command box 4.4.4.B. Given that the next computation starts from the invasion threshold of the predator the value of 0 for the initial predator density is correct. The value of `output2$bifpoints[4]`, however, representing the total biomass density of consumers vulnerable to predation is also 0, because it was produced by a call to `PSPMequi` with the option `c("envZE", "2")`, which forces this environment variable to equal 0. Since this is not appropriate as estimate for the environment variable in an equilibrium with predator, consumer and resource present, the value of the interaction variable `I[0][1]` (column 7 in `output2$bifpoints`) is used instead as initial estimate, which corresponds to the population integral representing the total biomass of small consumers that are vulnerable to predation (see code blocks 4.3.2.11 and 4.3.2.12). Hence, the vector `output2$bifpoints[c(1,2,3,7,5)]` is used as initial point for the computation shown in the next box:

Command box 4.4.4.C

```
> output3 <- PSPMequi("PNAS2002","EQ",output2$bifpoints[c(1,2,3,7,5)],-0.1,c(1,0,4E-4),NULL,NULL)

Dynamic library file PNAS2002equi.so is up-to-date

  2.53602314E-04  8.85690312E-06  0.00000000E+00  4.00801599E-06  2.33333540E-06
  2.52575824E-04  8.85867223E-06  9.53209678E-08  4.00801603E-06  2.36171847E-06
  2.51552996E-04  8.86045755E-06  1.90561304E-07  4.00801603E-06  2.39038298E-06
<...output lines suppressed in this box...>
  8.84797878E-05  1.31293570E-05  3.80288012E-05  4.00801603E-06  5.16666426E-05
```

```
  8.84988526E-05  1.32327343E-05  3.84539884E-05  4.00801603E-06  5.25658260E-05  **** LP      ****
  8.84988526E-05  1.32327343E-05  3.84539884E-05  4.00801603E-06  5.25658260E-05
<...output lines suppressed in this box...>
  3.89812802E-04  2.39609635E-04  1.32971757E-04  4.00801603E-06  2.92629903E-04
  3.97021837E-04  2.46474525E-04  1.33292505E-04  4.00801603E-06  2.93524789E-04
  4.04225759E-04  2.53351050E-04  1.33597449E-04  4.00801603E-06  2.94375934E-04

#
# Executing :
    PSPMequi("PNAS2002","EQ",c(0.000253602,8.8569E-06,0,4.00802E-06,2.33334E-06),-0.1,c(1,0,0.0004),NULL,NULL)
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  0.000253602    Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
#
# 1:           Rmax       2:E[0]       3:E[1]       4:E[2]       5:b[0]       10:pcgE[1] 11:R0[0]  12:RHS norm
#
> output3$curvepoints
                 V1           V2           V3           V4           V5 ..           V10      V11          V12
  [1,] 2.536023e-04 8.856903e-06 0.000000e+00 4.008016e-06 2.333335e-06 .. -8.589692e-11        1 5.165766e-08
  [2,] 2.525758e-04 8.858672e-06 9.532097e-08 4.008016e-06 2.361718e-06 .. -6.953758e-11        1 3.101426e-08
  [3,] 2.515530e-04 8.860458e-06 1.905613e-07 4.008016e-06 2.390383e-06 .. -4.268759e-11        1 2.153277e-08
<...output lines suppressed in this box...>
[437,] 3.898128e-04 2.396096e-04 1.329718e-04 4.008016e-06 2.926299e-04 .. -5.820885e-10        1 2.337764e-07
[438,] 3.970218e-04 2.464745e-04 1.332925e-04 4.008016e-06 2.935248e-04 .. -5.802998e-10        1 2.330581e-07
[439,] 4.042258e-04 2.533510e-04 1.335974e-04 4.008016e-06 2.943759e-04 .. -5.761841e-10        1 2.314052e-07
> output3$bifpoints
                 V1           V2           V3           V4           V5 ..           V10      V11          V12
  [1,] 8.849885e-05 1.323273e-05 3.845399e-05 4.008016e-06 5.256583e-05 .. -1.641421e-10        1 6.625094e-08
> output3$biftypes
[1] "LP"
```

In the output shown in the box above a large number of output lines have been suppressed and intermediate columns have been deleted as in the previous `R` output box, because the page width does not allow them to be shown completely. Consult the listing in your `R` console for the complete output of the commands executed. Otherwise the layout of the output and the output variables of the call to `PSPMequi` shown in box 4.4.4.C is the same as in `R` command box 4.4.4.B. The data of all computed equilibrium points making up the curve are contained in `output3$curvepoints` and the description of the computed curve in the variable `output3$curvedesc`. Issuing the command `cat(output3$curvedesc, sep="\n")` would lead to the textual output with information about the computation that is also produced by the `PSPMequi` function when finishing.

This last call to the `PSPMequi` function illustrates the detection of the last type of bifurcation that can occur in the dynamics of the PSPM, the saddle-node bifurcation. Starting from the predator invasion threshold the curve representing the predator-consumer-resource equilibrium first bends toward lower values of R_{max} reaching a minimum at $R_{max} = 8.8476 \cdot 10^{-5}$. The curve subsequently turns toward higher values of R_{max} again. The saddle-node bifurcation or limit point occurs at this minimum value of $R_{max} = 8.8476 \cdot 10^{-5}$. The data pertaining to this limit point is contained in the variable `output3$bifpoints`, whereas its description `"LP"` is stored in `output3$biftypes`. The limit point is the minimum value of R_{max} for which a predator-consumer-resource equilibrium occurs. It is hence also referred to as the predator persistence boundary.

The demo script called `"deRoosPersson"` uses `output3$curvepoints`, `output3$bifpoints` and `output3$biftypes` to draw the additional curves representing the predator-consumer-resource equilibrium in the bifurcation graphs that already showed the curves resulting from the previous 2 call to

`PSPMequi` (see `R` command box 4.4.4.A and 4.4.4.B).

The following 3 calls to `PSPMequi` that are executed by the demo script called `"deRoosPersson"`, illustrated in `R` command box 4.4.4.D, 4.4.4.E and 4.4.4.F, compute the location of the 3 detected bifurcation points, the branching point representing the invasion threshold of the consumer, the branching point of environment variable 2 representing the invasion threshold of the predator and the limit point representing the persistence threshold of the predator, as a function of two parameters: the value of R_{max} and the value of the consumer background mortality μ_b. These calls hence all use as 4th argument to the function the vector `c(1,0,4.0E-4,11,0,0.1)` indicating that the parameters with index 1 and 11 are to be varied (see code block 4.3.1.4 or 4.3.2.2) within the ranges 0 to $4.0 \cdot 10^{-4}$ and 0 to 0.1, respectively.

Computing the consumer invasion boundary as a function of R_{max} and μ_b starts from the data on the branching point stored in the output variable `output1$bifpoints`, which was detected during the computation of the trivial equilibrium without any consumers and predators (see `R` command box 4.4.4.A). The first two elements of this vector represent the value of R_{max} and the equilibrium resource density at the consumer invasion boundary. To complete the specification of the initial point of the computation the default value of consumer background mortality (0.01) is added. Hence, the initial point of the computation is `c(output3$bifpoints[1:2],0.01)`. The type of the computation, which is the second argument to the `PSPMequi` function, is now specified as `"BP"` as opposed to the value `"EQ"` that was used in all previous calls to `PSPMequi`.

At the consumer invasion boundary, the environment variables with index 1 and 2, representing the predator density and the total biomass density of small consumers vulnerable to predation, respectively, are both 0. The last argument of the call to `PSPMequi` shown in `R` command box 4.4.4.D hence equals the vector `c("envZE","1","envZE","2","popBP","0")`, which in addition to the zero equilibrium value for the environment variables also instructs the computational module that the transcritical bifurcation point that is computed occurs in the population with index 0.

Command box 4.4.4.D

```
> output4 <- PSPMequi("PNAS2002","BP",c(output1$bifpoints[1:2],0.01),0.05,c(1,0,4E-4,11,0,0.1),NULL,
                      c("envZE","1","envZE","2","popBP","0"))

Dynamic library file PNAS2002equi.so is up-to-date

  8.85690312E-06  8.85690312E-06  1.00000000E-02
  8.88483087E-06  8.88483087E-06  1.03112655E-02
  8.91358293E-06  8.91358293E-06  1.06066164E-02
<...output lines suppressed in this box...>
  3.94598924E-04  3.94598924E-04  8.27003296E-02
  3.98126241E-04  3.98126241E-04  8.27343996E-02
  4.01653833E-04  4.01653833E-04  8.27678949E-02

#
# Executing : PSPMequi("PNAS2002","BP",c(8.8569E-06,8.8569E-06,0.01),0.05,c(1,0,0.0004,11,0,0.1),NULL,
                       c("envZE", "1", "envZE", "2", "popBP", "0"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  8.8569E-06     Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
# Index of bifurcation parameter #2                          : 11
# Index of structured population with transcritical bifurcation: 0
#
```

```
#            1:Rmax       2:E[0]        3:E[1]       4:E[2]       5:b[0]      6:Mub  ..   13:RHS norm
#
> output4$curvepoints
                  V1           V2            V3           V4           V5         V6  ..           V13
  [1,] 8.856903e-06 8.856903e-06             0            0            0 0.01000000  ..  4.920353e-08
  [2,] 8.884831e-06 8.884831e-06             0            0            0 0.01031127  ..  1.327406e-08
  [3,] 8.913583e-06 8.913583e-06             0            0            0 0.01060662  ..  1.515673e-08
<...output lines suppressed in this box...>
[511,] 3.945989e-04 3.945989e-04             0            0            0 0.08270033  ..  9.558855e-10
[512,] 3.981262e-04 3.981262e-04             0            0            0 0.08273440  ..  9.616150e-10
[513,] 4.016538e-04 4.016538e-04             0            0            0 0.08276789  ..  9.642249e-10
```

(Once again, consult your R console for a complete listing of the output of the commands shown above as parts of this output is deleted for the sake of brevity and layout).

The output list of the PSPMequi function now only contains 2 member elements, the data matrix output4$curvepoints containing all information about the points making up the computed curve and the description variable output4$curvedesc, whose contents can be shown by executing cat(output4$curvedesc,sep="\n") and is also printed by the PSPMequi function to the R console on exit. The output list does not include the two additional output elements containing information about bifurcations, since no detection of such special points is carried out during computations of transcritical bifurcation boundaries. As shown in the R command box above the output now contains an additional column, which follows the values of the birth rate of the structured population in equilibrium. This column contains the value of the second bifurcation parameter, which corresponds to μ_b in the PNAS model. The demo script called "deRoosPersson" subsequently uses the first and and the sixth column of output4$curvepoints to create a graph with R_{max} (first column) on the x-axis and μ_b (sixth column) on the y-axis, showing the regions of parameter space for which persistence of the consumer population is and is not possible.

Computing the predator invasion boundary as a function of R_{max} and μ_b starts from the data on the branching point stored in the output variable output2$bifpoints, which was detected during the computation of the consumer-resource equilibrium without any predators (see R command box 4.4.4.B). The first two elements of this vector represent the value of R_{max} and the equilibrium resource density at the predator invasion boundary, while the fifth element of output2$bifpoints represents the birth rate of the structured consumer population in this equilibrium (see the listing of output2$bifpoints in R command box 4.4.4.B). To complete the specification of the initial point of the computation the default value of consumer background mortality (0.01) is added. Hence, the initial point of the computation is c(output2$bifpoints[c(1,2,5)],0.01). The type of the computation, which is the second argument to the PSPMequi function, is now specified as "BPE", indicating that a transcritical bifurcation curve in an environment variable is to be computed as a function of 2 model parameters.

The environment variables with index 2, representing the total biomass density of small consumers vulnerable to predation does not affect the predator invasion boundary, as it only influences the predation mortality of small consumers (see code block 4.3.2.10). This environment variable can hence be assumed to equal 0 and its equilibrium condition can be ignored by passing the appropriate option pair to the PSPMequi function. Furthermore, the option vector should instruct the computational module to compute the transcritical bifurcation curve for the environment variable with index 1. The last argument of the call to PSPMequi shown in R command box 4.4.4.E therefore equals the vector c("envBP","1","envZE","2"). Notice that compared to the previous call to PSPMequi, this option array does not contain any specific instructions concerning the structured consumer population.

Command box 4.4.4.E

```
> output5 <- PSPMequi("PNAS2002","BPE",c(output2$bifpoints[c(1,2,5)],0.01), 0.1,c(1,0,4E-4,11,0,0.1),NULL,
                      c("envZE","2","envBP","1"))

Dynamic library file PNAS2002equi.so is up-to-date
```

```
5
  2.53602314E-04  8.85690312E-06  2.33333540E-06  1.00000000E-02
  2.44422274E-04  8.88032498E-06  2.35250524E-06  1.02628274E-02
  2.35413020E-04  8.90710825E-06  2.37324896E-06  1.05420330E-02
<...output lines suppressed in this box...>
10  3.88476406E-04  2.61758185E-04  1.08055050E-05  8.07841851E-02
  3.95419276E-04  2.68571865E-04  1.08238254E-05  8.09262046E-02
  4.02376220E-04  2.75405272E-04  1.08413352E-05  8.10619333E-02

#
15 # Executing :
    PSPMequi("PNAS2002","BPE",c(0.000253602,8.8569E-06,2.33334E-06,0.01),-0.1,c(1,0,0.0004,11,0,0.1),NULL,
            c("envZE","2","envBP","1"))
#
# Parameter values  :
#
20 #   Rho       :  0.1            Rmax      :  0.000253602    Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
25 #   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
# Index of bifurcation parameter #2                          : 11
# Index of environment variable with transcritical bifurcation : 1
30 #
#            1:Rmax       2:E[0]        3:E[1]       4:E[2]       5:b[0]      6:Mub  ..   13:RHS norm
#
> output5$curvepoints
                 V1           V2            V3           V4           V5          V6  ..           V13
35  [1,] 0.0002536023 8.856903e-06             0            0 2.333335e-06 0.01000000  ..  5.082990e-08
  [2,] 0.0002444223 8.880325e-06             0            0 2.352505e-06 0.01026283  ..  1.784247e-08
  [3,] 0.0002354130 8.907108e-06             0            0 2.373249e-06 0.01054203  ..  2.365370e-08
<...output lines suppressed in this box...>
[214,] 0.0003884764 2.617582e-04             0            0 1.080551e-05 0.08078419  ..  1.457065e-09
40 [215,] 0.0003954193 2.685719e-04             0            0 1.082383e-05 0.08092620  ..  1.524866e-09
[216,] 0.0004023762 2.754053e-04             0            0 1.084134e-05 0.08106193  ..  1.587493e-09
```

(Once again, consult your R console for a complete listing of the output of the commands shown above as parts of this output is deleted for the sake of brevity and layout).

The R command box above shows that the output of the PSPMequi function has in this case a similar layout as when computing the consumer invasion boundary (R command box 4.4.4.D). The data about the points making up the computed curve are contained in output5$curvepoints, whereas output5$curvedesc contains the description of the computations. These are the only two elements of the output list for computations involving two variable parameters, as was the case for the continuation of the "BP" curve. The demo script called "deRoosPersson" again uses the first and and the sixth column of output5$curvepoints to create a graph with R_{max} (first column) on the x-axis and μ_b (sixth column) on the y-axis, showing the regions of parameter space for which invasion of the consumer-resource equilibrium by the predator is possible or not.

The final analysis step to be performed is to compute the location of the limit point in the predator-consumer-resource equilibrium curve as a function of R_{max} and μ_b. This computation starts from the data on the limit point stored in the output variable output3$bifpoints, which was detected during the computation of the predator-consumer-resource equilibrium (see R command box 4.4.4.C). The first five elements of this vector represent the value of R_{max}, the equilibrium resource density, the equilibrium predator density, the equilibrium biomass density of small consumers vulnerable to predation and the equilibrium birth rate of the consumer population. To complete the specification of the initial point of the computation the default value of consumer background mortality (0.01) is added. Hence, the initial point of the computation is specified as c(output3$bifpoints[1:5],0.01). The type of the computation, which is the second argument to the PSPMequi function, is now specified

as "LP", indicating that a saddle-node bifurcation curve is to be computed as a function of 2 model parameters.

For this computation the option vector is left undefined (NULL), because all variables influence the location of the limit point and hence none of the quantities are characterized by a zero equilibrium state.

Command box 4.4.4.F

```
> output6 <- PSPMequi("PNAS2002","LP",c(output3$bifpoints[1:5],0.01),0.05,c(1,0,4E-4,11,0,0.1),NULL,NULL)

Dynamic library file PNAS2002equi.so is up-to-date

  8.84988526E-05  1.31478259E-05  3.80697042E-05  4.00801603E-06  5.17387220E-05  1.00133536E-02
  8.86520573E-05  1.31876853E-05  3.79221830E-05  4.00801603E-06  5.13036116E-05  1.01310924E-02
  8.88061573E-05  1.32276419E-05  3.77736423E-05  4.00801603E-06  5.08693956E-05  1.02496461E-02
<...output lines suppressed in this box...>
  1.12749308E-04  1.70600662E-05  5.08576521E-08  4.00801603E-06  4.95866499E-06  3.44777934E-02
  1.12762300E-04  1.70588531E-05  3.89902246E-09  4.00801603E-06  4.94169593E-06  3.45006828E-02
  1.12775266E-04  1.70576306E-05 -4.30752282E-08  4.00801603E-06  4.92476988E-06  3.45235657E-02

#
# Executing : PSPMequi("PNAS2002","LP",c(8.84989E-05,1.32327E-05,3.8454E-05,4.00802E-06,5.25658E-05,0.01),0.05,
                      c(1,0,0.0004,11,0,0.1),NULL,NULL)
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  8.84989E-05    Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index of bifurcation parameter #1                          : 1
# Index of bifurcation parameter #2                          : 11
#
#            1:Rmax       2:E[0]        3:E[1]       4:E[2]       5:b[0]      6:Mub  ..   13:RHS norm
#
> output6$curvepoints
                 V1           V2            V3           V4           V5          V6  ..           V13
  [1,] 8.849885e-05 1.314783e-05  3.806970e-05 4.008016e-06 5.173872e-05 0.01001335  ..  1.618225e-10
  [2,] 8.865206e-05 1.318769e-05  3.792218e-05 4.008016e-06 5.130361e-05 0.01013109  ..  1.958108e-11
  [3,] 8.880616e-05 1.322764e-05  3.777364e-05 4.008016e-06 5.086940e-05 0.01024965  ..  1.912080e-11
<...output lines suppressed in this box...>
[292,] 1.127493e-04 1.706007e-05  5.085765e-08 4.008016e-06 4.958665e-06 0.03447779  ..  6.794966e-12
[293,] 1.127623e-04 1.705885e-05  3.899022e-09 4.008016e-06 4.941696e-06 0.03450068  ..  5.837794e-12
[294,] 1.127753e-04 1.705763e-05 -4.307523e-08 4.008016e-06 4.924770e-06 0.03452357   .. 2.876338e-12
```

(Once again, consult your R console for a complete listing of the output of the commands shown above as parts of this output is deleted for the sake of brevity and layout).

The output of the PSPMequi function in this last call has a similar layout as in the previous two calls to compute consumer and predator invasion boundary (R command box 4.4.4.D and 4.4.4.E). The data about the points making up the computed curve are stored in output6$curvepoints, whereas the description of the computations is stored in output6$curvedesc. These are, as before, the only two elements of the output list output6. The demo script called "deRoosPersson" again uses the first and and the sixth column of output6$curvepoints to create a graph with R_{max} (first column) on the x-axis and μ_b (sixth column) on the y-axis, showing the regions of parameter space for which a predator-consumer-resource equilibrium occurs or not and hence for which predators can persist.

4.4.5 Output files generated by the PSPMequi function

The computational module that is produced by the PSPMequi function generates 4 output files in case of a one-parameter bifurcation (continuation type "EQ") and 3 output files in other cases (continuation types "BP", "BPE", "LP", "ESS" and "PIP"). The name of these files is always of the form <Modelname>-<Type>-<NNNN>.<ext>, in which <Modelname> is the same as the name of the file specifying the model excluding its '.R' or '.h' extension, <Type> refers to the type of the continuation performed (either EQ, BP, BPE, LP, ESS or PIP) and <NNNN> is a 4-digit number that is unique for the current computation and .<ext> is the extension, which can be either .bif, .err, .csb or .out. The unique number distinguishes the same types of curve computations for the same model from each other. The number is obtained by considering for a specific type of continuation ("BP", "BPE", "EQ", "LP", "ESS" or "PIP") increasing values of <NNNN> (i.e., 0000, 0001, 0002 and so forth) and testing whether result files with the particular index are already present. The program uses the first value of <NNNN> that is not in use.

Hence, the call of the PSPMequi function for the PNAS model, as shown in R command box 4.4.4.A generates the output files PNAS2002-EQ-0000.bif, PNAS2002-EQ-0000.err, PNAS2002-EQ-0000.out and PNAS2002-EQ-0000.csb, the following call as shown in R command box 4.4.4.B generates the output files PNAS2002-EQ-0001.bif, PNAS2002-EQ-0001.err, PNAS2002-EQ-0001.out and PNAS2002-EQ-0001.csb, while the last computation of an equilibrium curve, as shown in R command box 4.4.4.C, generates the output files PNAS2002-EQ-0002.bif, PNAS2002-EQ-0002.err, PNAS2002-EQ-0002.out and PNAS2002-EQ-0002.csb. The computations of the consumer invasion, predator invasion and predator persistence boundary (see R command box 4.4.4.D, 4.4.4.E and 4.4.4.F) each generate only 3 output files, called PNAS2002-<Type>-0000.err, PNAS2002-<Type>-0000.out and PNAS2002-<Type>-0000.csb with <Type> equal to BP, BPE and LP in case of the consumer invasion, the predator invasion and the predator persistence boundary, respectively.

The file called <Modelname>-<Type>-<NNNN>.err that is generated during the computations of curves contains information about the numerical progress of the computations. It reports details on the steps taken during the Newton iteration, the convergence to the solution, as well as information about the steps taken along the curve that is being computed. This file can be informative in case the computation of a particular curve stops for unknown reasons, but is otherwise of little use.

The output file called <Modelname>-<Type>-<NNNN>.out contains the same information as in the member elements curvepoints and curvedesc of the output list returned by the PSPMequi function (see the help page on PSPMequi using ?PSPMequi). The first lines of this file all start with a '#' sign and contain the information about the run performed, which is also contained in curvedesc and can be listed by the statement cat(output$curvedesc,sep="\n"). Following this descriptive header the file contains columns with computational results that are also contained in the variable curvepoints (see, for example, R command box 4.4.4.B). In fact, the first two output elements curvepoints and curvedesc are generated by reading the contents of the file <Modelname>-<Type>-<NNNN>.out from disk after the computations have ended, storing all lines that start with a '#' sign into curvedesc, while storing the information on all other lines into the data matrix curvepoints.

Similarly, the output file called <Modelname>-<Type>-<NNNN>.bif, which is only generated during the computation of an equilibrium curve (type "EQ"), contains the same information as is contained in the last two elements of the output list, called bifpoints and biftypes, returned by the PSPMequi function (see the help page on PSPMequi using ?PSPMequi). Each row in the file <Modelname>-<Type>-<NNNN>.bif pertains to a single detected bifurcation point. A row starts with the numerical data that characterizes the bifurcation point, which are exactly the same columns of data as stored in the file <Modelname>-<Type>-<NNNN>.out. Appended to the numerical data is a string of the form *** <Type> ***, where <Type> can be, for example, BP #0, BP #0, LP or CSS #0. The numerical data that form the first part of each row are stored by the PSPMequi function into the list element bifpoints, which hence has as many columns as there are in the output list element curvepoints and as many

rows as there bifurcation points occurring in the computed equilibrium curve. The strings representing the type of bifurcation point are stored by the `PSPMequi` function into `biftypes`, which hence has as many elements as there are bifurcation points.

The file called `<Modelname>-<Type>-<NNNN>.csb` contains for every curve point that has been computed information on the parameters, for which the point has been computed, the equilibrium values of all environment variables and the stable distribution of all structured populations in the model. This is a binary file, the content of which can be accessed from `R` using the function `csbread`. For example, the file `PNAS2002-EQ-0002.csb` is generated by the invocation of the `PSPMequi` function for the PNAS model shown in `R` command box 4.4.4.C. Its contents can be listed by:

Command box 4.4.5.A

```
> csbread("PNAS2002-EQ-0002.csb")

States in file PNAS2002-EQ-0002.csb:

    1: State-2.536023E-04
    2: State-2.525758E-04
    3: State-2.515530E-04
<...output lines suppressed in this box...>
  437: State-3.898128E-04
  438: State-3.970218E-04
  439: State-4.042258E-04
```

The population state called `State_4_042258E_04` pertains to the parameter value $R_{max} = 4.042258 \cdot 10^{-4}$ as its name suggests. Its contents can be read into the workspace by issuing the command `csbread("PNAS2002-EQ-0002.csb",439)` or equivalently `csbread("PNAS2002-EQ-0002.csb","State-4.042258E-04")`.

Loading this state into the `R` workspace reveals it to be a list containing various arrays of numbers, as shown in the following box:

Command box 4.4.5.B

```
> popstate <- csbread("PNAS2002-EQ-0002.csb", "State-4.042258E-04")
> popstate
$BifPars
[1] 0.0004042258

$Parameters
 [1] 1.000000e-01 4.042258e-04 7.000000e+00 2.700000e+01 1.100000e+02 3.000000e+02 1.500000e-05 .....
[15] 5.000000e-01 1.000000e-02

$Environment
[1] 2.533511e-04 1.335974e-04 4.008016e-06

$Pop00_BirthStates
     Istate00 Istate01
[1,]        0        7

$Pop00
            Density    Istate00   Istate01
  [1,] 4.349476e-04    1.475004   9.423334
  [2,] 7.156099e-07   18.427094  35.856199
  [3,] 6.318860e-07   30.803859  53.560944
<...output lines suppressed in this box...>
 [98,] 4.965604e-12 1206.196792 283.032172
 [99,] 4.387718e-12 1218.569350 283.046392
[100,] 3.877086e-12 1230.941907 283.059594
```

The first element of the list (called `$BifPars`) representing the population state `State-4.042258E-04` is the value of the bifurcation parameter(s) for this particular state. The second element, an array called `$Parameters` (not completely displayed in the box above because of space restrictions), contains

the values of all the model parameters characterizing this particular equilibrium state, while the third member of the list contains the equilibrium values of all environment variables. The two subsequent arrays in the list characterize the stable population distribution, of which the first (called `$Pop00_BirthStates`) specifies the state at birth of the individuals. The other (called `$Pop00`) is a two-dimensional array characterizing the population distribution in equilibrium with the first column `$Pop00[,1]` representing the density profile of the equilibrium population and the subsequent columns `$Pop00[,2]` and `$Pop00[,3]` representing the average values of the individual state variables with index 0 and 1 in the model (corresponding to individual age and length in the PNAS model), as shown in the `R` command box above. If individuals are characterised by more than two individual state variables, the values of these follow in additional columns of the two-dimensional array `$Pop00`. The `R` command box above also illustrates that the dimension of the array `$Pop00` indicates that the population is represented by 100 cohorts of individuals (see section 9.3 for the option to change this number). The number of individuals in cohort i is given by the array element `$Pop00[`i`,1]`, while the average value of the individual state variable with index 0 and 1 (average age and average length in the PNAS model) are given by `$Pop00[`i`,2]` and `$Pop00[`i`,3]`, respectively.

Chapter 5

Simulating ecological dynamics

The ecological dynamics of a PSPM can be simulated using the *Escalato Boxcar Train* (`EBT` in short). The `EBT` is a numerical method especially designed for integrating the partial differential equations that are the mathematical representations of PSPMs (De Roos 1988, De Roos, Diekmann, and Metz (1992)). The `EBT` method differs from standard numerical integration methods for partial differential equations, as its design is inspired by the biological underpinning of PSPMs, rather than by their mathematical expressions in terms of partial differential equations. A recent comparison of 4 different methods for the numerical integration of size-structured population models therefore concluded that the `EBT` method performs overall best and is the only method that can be straightforwardly extended to PSPMs with more than a single individual state variable (Zhang, Dieckmann, and Brännström 2017).

The EBTtool is a software package that implements the *Escalato Boxcar Train* and provides a graphical user interface for its operation. This software package has been around already for quite some years. The EBTtool is itself implemented in `C` and also requires that a particular PSPM also is implemented in `C`. The current `PSPManalysis` package includes a function `PSPMecodyn` that is a trimmed-down version of the EBTtool. This trimmed-down version does not include the graphical user interface and is limited to the type of models that can be implemented using the model specificaton templates of the `PSPManalysis` package (see the discussion in sections 4.3.1 and 4.3.2). For example, whereas pulsed reproduction is easily implemented and dealt with in the EBTtool, the function `PSPMecodyn` can not handle it as reproduction is assumed to be continuous. Nevertheless, `PSPMecodyn` is useful to study the ecological dynamics of the models that can be analyzed with the `PSPManalysis` package and works with exactly the same model implementation file that is used for all other computations as well. Notice, however, that the PSPM should be non-linear and hence include environmental state variables as well as feedback of the population on these environmental state variables. The `Medfly` model that was discussed in chapter 3 in the context of demographic analysis of PSPMs can hence not be studied using `PSPMecodyn`.

> Even though the EBTtool can not be used to numerically integrate a model that is implemented in the `PSPManalysis` template, it might nonetheless be a good idea to download and install this software as it provides a useful graphical user interface to explore the results of numerical integrations. In particular, it uses the same type of data files as produced by the `PSPManalysis` package (output files with extensions `.out` and `.csb`; see section 5.3 below) and is particularly useful to analyze the changes in the population distribution over time. The EBTtool moreover implements methods to output sequences of these population states to an animated GIF file that can be used for presentations.

5.1 Arguments and output of the PSPMecodyn function

The use of the PSPMecodyn function will be illustrated with the PNAS model as described in section 4.2 and analysed in section 4.4. A demo script "deRoosPerssonDynamics" is included with the PSPManalysis package that carries out numerical integration of the PNAS model, starting from three different initial conditions.

The general call to the PSPMecodyn function is shown in the command box below.

```
> output <- PSPMecodyn(modelname = NULL, startstate = NULL, timepars = NULL, bifpars = NULL,
                       parameters = NULL, options = NULL, clean = FALSE, force = FALSE, debug = FALSE)
```

The obligatory and optional arguments to the PSPMecodyn function are the following:

1. The first, obligatory argument to the function PSPMecodyn is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension '.R' or '.h' as part of the file name, the PSPMecodyn function will automatically try to locate the appropriate file, checking first for a file implemented in C (with an extension '.h') and subsequently for a file implemented in R (with an extension '.R'). If both a file with an extension '.h' and a file with an extension '.R' are found, the program will use the first one. The program can be forced to use the file with an extension '.R' by including the extension explicitly as part of the filename. The R-command to simulate the dynamics of the PNAS model that will be used as illustrations below will therefore all take "PNAS2002" as their first argument. If the file specifying the PSPM can not be found in the current directory, the PSPMequi function will ask the user to search in the package directory for a model file with the specified name.

2. The second, obligatory argument is the initial condition of the computation. This initial point should be a list that has the same structure as the list returned by the function csbread (see, for example, section 4.4.5). The list should at least include an element $Environment and elements $Pop00, $Pop01, $Pop02, etc., for as many structured populations are accounted for in the model. The element $Environment should be a vector with initial values of the environmental state variables for the simulation, while the elements $Pop00, $Pop01, $Pop02, etc., should be matrices with a number of columns that is exactly 1 larger than the number of individual state variables in the model. Each row of these matrices should specify the initial state of a cohort of individuals in the particular population. The first column of each of these matrices, i.e. $Pop00[$i$,1], should specify the number of individuals in cohort i, while the average value of the individual state variable with index 0 and 1 (average age and average length in the PNAS model) have to be specified in the second and third column of each matrix, $Pop00[$i$,2] and $Pop00[$i$,3], respectively. Naturally, if individuals are characterized by more than two individual state variables, the values of these have to follow in additional columns. In general, there are 3 different methods to construct an initial state for the function PSPMecodyn: (1) the initial state can be a population state that results from an equilibrium computation with PSPMequi; (2) the initial state can be generated using the function PSPMind that computes the individual life history under a specific set of environmental conditions (see chapter 8); and, (3) the initial state can be constructed by the user with list-construction commands in R. These 3 different methods will be discussed in more detail in section 5.2.

   If the list in the second argument contains an element $Parameters and this vector is of the appropriate length, this vector will be used as values for the model parameter, for which to carry out the integration. This vector should have the same length as the number of parameters in the model (the length of the vector DefaultParameters in R, or the value of PARAMETER_NR in C). When of this length the values will replace the default values of the parameters that are listed in the model specification file. However, if a vector of parameter values is included as fifth argument

in the call to the function PSPMecodyn, the latter supersede any values that may be specified in the initial state list.

In principle, the list that is given as the second argument to the function PSPMecodyn should also include elements called $Pop00_BirthStates, $Pop01_BirthStates, $Pop02_BirthStates, etc., that specify the state at birth of the individuals in the different populations as the state at birth may influence the life history dynamics. These elements are automatically included when the initial state is produced by either a call to the function PSPMequi or to the function PSPMind. They consist of a vector (in case of a single state at birth) or a two-dimensional matrix of values (in case of multiple states at birth) with the values of the individual state variables with which individuals can be born, each row representing a state at birth and each column an individual state variable. If their are multiple states at birth the initial state list should contain elements $Pop00_Bstate00, $Pop00_Bstate01, $Pop00_Bstate02, etc., specifying the subpopulations originating from each of the states at birth (see command box 9.1.1.1.B in section 9.1). If these elements are missing, however, the function PSPMecodyn will automatically generate valid individual states at birth through calls to the functions SetBirthStates() and StateAtBirth() (see sections 4.3.2.4 and 4.3.2.5) or their appropriate counterparts when the model is implemented in R.

3. The third, obligatory argument to the PSPMecodyn function is a row vector consisting of 4 elements:

   c(<*cohort limit*>,<*output interval*>,<*state output interval*>,<*maximum time*>)

   In this vector <*cohort limit*> refers to the time interval during which a new cohort of individuals is formed. The EBT method is based on the idea to collect all individuals that are born within a interval Δt into a single cohort of the population (De Roos 1988, De Roos, Diekmann, and Metz (1992)). The element <*cohort limit*> sets the value of this time interval Δt. Smaller values of Δt will mean that the computed model trajectories are more accurate, but will also slow down the computation as the populations will include more cohorts and hence the system of equations to integrate is larger. The vector element <*output interval*> defines the time interval between data output to the output file with extension .out (see section 5.3 below), while the vector element <*state output interval*> similarly defines the time interval between output of the entire population state to the output file with extension .csb (see section 5.3 below). Finally, the vector element <*maximum time*> sets the maximum time at which to halt the numerical integration.

4. The fourth, optional argument to the PSPMecodyn function can be used to carry out bifurcation analysis of the model dynamics, in case the model predicts non-equilibrium dynamics. An easy way to create a bifurcation graph is to carry out a large number of numerical integrations of the model equations, each with a slightly different value for the bifurcation parameter. This approach, however, has its problems, as the computed outcome will depend on the choice of the initial values for the model variables. One way to circumvent the problems associated with the initial value of model variables is to carry out only a single numerical integration of the model equations but over a very long time period, in which the entire integration period is subdivided into intervals during which the value of the bifurcation parameter is constant, while from one interval to the next the value of the bifurcation parameter is increased or decreased by a small amount. In this way, the range of values of the bifurcation parameter is scanned either from low to high or vice versa. This stepwise increase or decrease of the bifurcation parameter implies that the final values of the model variables obtained for a particular value of the bifurcation parameter are used as initial values of the model variables for the subsequent parameter value.

The advantage of this approach can best be explained in the context of stable model equilibria, but it works equally well in case of non-equilibrium dynamics. Consider that after a numerical integration over a sufficiently long time interval for a particular value of the bifurcation parameter all model variables have ended up close their equilibrium value. These final values of the model variables will also be close to their equilibrium value for a setting of the bifurcation parameter that is slightly larger or smaller, provided that the particular equilibrium still exists for this new parameter value. Hence, adopting these final values of the model variables as initial values for a subsequent integration with a slightly different bifurcation parameter value ensures that we continue to follow the curve of a particular model equilibrium as a function of the bifurcation parameter as long as the equilibrium exists and is stable. Only when the equilibrium becomes unstable or does not occur at all any more for the new value of the bifurcation parameter, will the model variables approach an entirely different equilibrium or a different type of dynamics, such as a limit cycle. By scanning a particular interval of the bifurcation parameter with increasing as well as decreasing parameter values in most cases also reveals the co-occurrence of alternative stable equilibria or alternative types of stable dynamics, such as the co-occurrence of different types of limit cycles, for the same value of the bifurcation parameter.

If specified, this argument has to be a vector with 6 elements:

`c(`<*index*>,<*start*>,<*step*>,<*stop*>,<*output period*>,<*state output period*>`)`

In this vector <*index*> indicates the index of the bifurcation parameter to vary, <*start*> indicates the starting value of the bifurcation parameter, <*step*> the step size in the bifurcation parameter, <*stop*> the final value of the bifurcation parameter, <*output period*> the time interval at the end of each bifurcation interval during which data output is written to the file with `.out` (see section 5.3 below), and <*state output period*> determines the time interval at the end of each bifurcation interval during which output of the entire population state is written to the file with `.csb` (see section 5.3 below).

5. The fifth, optional argument of the `PSPMecodyn` function is a (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (the length of the vector `DefaultParameters` in `R`, or the value of `PARAMETER_NR` in `C`). When of this length the values will replace the default values of the parameters that are listed in the model specification file. If the array used for this sixth argument is not of the correct length or when it is not specified at all, it will simply be ignored.

6. The sixth, optional argument of the `PSPMecodyn` function is a (row) vector of string elements, containing possible options that modify the behavior of the computational module. Only two options are possible and both require a value and hence occur as a pair of option name and option value. The option argument is therefore either of the form:

   `c("`*name 1*`", "`*value 1*`")`

   or of the form:

   `c("`*name 1*`", "`*value 1*`","`*name 2*`", "`*value 2*`")`

   The options can be specified in any order, but the option value should always immediately follow after the option name. Possible options are:

   i. Option pair `c("info", "`*i*`")`: This option modifies the output of the DOPRI5 integration method to the output file with extension `.err` (see section 5.3 below). The value of *i* can be set equal to 1, 2, 3 or 4. The default behavior of the function `PSPMecodyn` is equivalent to *i* equal to 0, in which case no output at all is produced in the file with extension `.err`. With higher values of *i* more information is produced detailing the performance of the integration

method (i.e. step sizes used, number of succesfull and unsuccesfull integration steps, etc.). This output is only useful in case problems occur during the numerical integration of the model.

ii. Option pair `c("report", "`*i*`")`: This option, which should have a positive value *i*, determines the time interval between consecutive lines of data output to the console. The default behavior of the function `PSPMecodyn` is to write a line of output to the console whenever a new cohort of individuals is constructed. However, this may result in a lot of output. By choosing a higher value for this option the program can be forced to write to the console only every 10th time that a cohort is constructed. Notice that this option does not affect the freequency with which output is written to the output file with extension `.out` (see section 5.3 below).

Three other optional arguments can be passed to the `PSPMecodyn` function: `clean`, `force` and `debug`. These are all boolean arguments that hence have to be passed to the `PSPMecodyn` function as <*option name*>`=TRUE` or <*option name*>`=FALSE`, the latter being the default value of all options (Specifying these options as argument is hence only useful when setting them equal to `TRUE`). Unlike the previous arguments, which all modify the computations to be performed, these options modify the behavior of the `PSPMecodyn` function itself, in particular the compilation of the model specific file into a dynamic library module that can be executed from `R`. Also unlike all the previous arguments that can be passed, these arguments can be passed in any order and at any position, the `PSPMecodyn` function will filter these 3 optional arguments from the argument list before passing the filtered argument list to the computational routine.

- Option `clean`: When `clean=TRUE` is passed as argument, this argument instructs the `PSPMecodyn` function to delete all result files that have been generated during previous calculations with the model. These result files have names of the form `<Modelname>-<Type>-<NNNN>.err`, `<Modelname>-<Type>-<NNNN>.csb` and `<Modelname>-<Type>-<NNNN>.out`, in which `<Modelname>` refers to the name of the model, `<Type>` refers to the type of computation that has been performed, which in the case of `PSPMecodyn` equals `ECODYN`, and `<NNNN>` is a unique number that distinguishes consecutive computations of the same type of curve with the same model. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function `PSPMclean()` to delete all `.bif`, `.err`, `.csb` and `.out` files and/or all executable files that are present in the current working directory.

- Option `force`: When `force=TRUE` is passed as argument, it instructs the `PSPMecodyn` function to force re-compilation of the model specific file into a dynamic library module that can be executed by `R`. This option will usually not be needed by normal users, as the `PSPMecodyn` function automatically recompiles the computational module when the model specific file with an `'.h'` or `'.h'` extension is more recently changed than the compiled dynamic library file. However, if for some unclear reason this automatic recompilation fails, the `force` option can be used to initiate re-compilation.

- Option `debug`: When `debug=TRUE` is passed as argument, it instructs the `PSPMecodyn` function to turn on debugging flags while compiling the model specific file into a dynamic library module. This option can be useful to detect programming mistakes in the model-specific file that are otherwise hard to track down. The downside is that depending on the version of `R` that is used, turning on debugging flags during compilation may generate a lot of output, including warnings about standard files of the operating system that are perfectly correct. It is hence not so easy to spot among all these messages the warnings that relate to the model-specific code that has been implemented.

The computational module generates on execution a single list object as output with 2 member elements (see the help page on `PSPMecodyn` using `?PSPMecodyn`). The first element of the output list, `output$curvepoints` contains the numerical information of the points along the computed curve. This variable `output$curvepoints` is a matrix, in which each row represents one solution point along the curve. The columns contain the time value, the value of all environment variables, the value for the birth rate of all structured populations in the problem, followed by the value of all interaction variables defined in the routine `Impact()`.

The second member element of the output list, `output$curvedesc`, contains the description of the executed calculation, which includes the command-line that is used for the invocation of the computational routine, the values of all parameters used for the current computation and a header line indicating the meaning of all the output variables produced by the computational module. This textual information is also printed to the `R` console at the end of calculations. In fact, the `PSPMecodyn` function prints its report on the calculations by execution of the statement `cat(output$curvedesc, sep='\n')`.

5.2 An example session using the `PSPMecodyn` function

The demo script `"deRoosPerssonDynamics"` illustrates the use of the `PSPMecodyn` function by simulating the ecological dynamics of the PNAS model as presented in section 4.2, starting from 3 different initial conditions. The first numerical integration starts from an equilibrium state computed with the function `PSPMequi`. The particular calls to the `PSPMequi` and `PSPMecodyn` functions are shown in the following `R` command box:

Command box 5.2.A

```
> output <- PSPMequi("PNAS2002", "EQ", c(3.0E-04, 1.561E-04, 1.270E-04, 4.008E-06, 2.761E-04), 0.1, c(1, 0, 1),
    options = c("single"), clean = TRUE, force = TRUE)

<...compilation output lines suppressed in this box...>

  3.00000000E-04, 1.56127570E-04, 1.27032702E-04, 4.00801603E-06, 2.76129173E-04

#
# Executing : PSPMequi("PNAS2002", "EQ", c(0.0003, 0.0001561, 0.000127, 4.008E-06, 0.0002761), 0.1, c(1, 0, 1),
    NULL, c("single"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  0.0003         Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
# Index and name of bifurcation parameter #1                  :  1 (Rmax)
#
#         1:Rmax         2:E[ 0]         3:E[ 1]         4:E[ 2]         5:b[ 0]     6:I[ 0][ 0]     7:I[ 0][ 1]
        8:I[ 0][ 2]     9:I[ 0][ 3]     10:pcgE[ 1]       11:R0[ 0]     12:RHS norm
#
> initstate <- csbread("PNAS2002-EQ-0000.csb", 1)
> output1 <- PSPMecodyn("PNAS2002", initstate, c(1, 1, 10, 1000), options = c("report", "50"), clean = TRUE,
    force = TRUE)

<...compilation output lines suppressed in this box...>

       0.00, 1.56127570E-04, 1.27032702E-04, 3.26244320E-06
      50.00, 1.57798334E-04, 1.26716011E-04, 3.96498271E-06
     100.00, 1.59593562E-04, 1.26024893E-04, 3.92329007E-06
<...output lines suppressed in this box...>
```

```
      900.00, 1.56843463E-04, 1.27713985E-04, 3.95446742E-06
      950.00, 1.58701063E-04, 1.26750658E-04, 3.93952452E-06
     1000.00, 1.59249607E-04, 1.25798353E-04, 3.95558086E-06

RUN PNAS2002-ECODYN-0000 COMPLETED at T = 1000.00:
** Program terminated. Normal closure of output files succeeded.          **

#
# Executing : PSPMecodyn("PNAS2002", <Initial state>, c(1, 1, 10, 1000), NULL, NULL, c("report", "50"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  0.0003         Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
#   Cohort cycle time interval                                          : 1.0
#   Output time interval                                                : 1.0
#   Complete state output interval                                      : 10.0
#   Maximum integration time                                            : 1000.0
#
#
#     1:Time       2:E[0]       3:E[1]       4:E[2]       5:b[0] ..    7:I[0][1]    8:I[0][2]    9:I[0][3]
> output1$curvepoints
          V1           V2           V3           V4           V5 ..           V7           V8           V9
   [1,]    0 0.0001561276 0.0001270327 3.262443e-06 0.0000000000 .. 3.262443e-06 1.414324e-05 0.0001710999
   [2,]    1 0.0001562952 0.0001268490 3.571646e-06 0.0002817937 .. 3.571646e-06 9.659792e-06 0.0001755738
   [3,]    2 0.0001563517 0.0001267466 3.775143e-06 0.0002807738 .. 3.775143e-06 1.000305e-05 0.0001752195
<...output lines suppressed in this box...>
 [999,]  998 0.0001592588 0.0001258301 3.954380e-06 0.0002686477 .. 3.954380e-06 1.157543e-05 0.0001699083
[1000,]  999 0.0001592545 0.0001258141 3.954975e-06 0.0002686223 .. 3.954975e-06 1.158832e-05 0.0001698792
[1001,] 1000 0.0001592496 0.0001257984 3.955581e-06 0.0002685985 .. 3.955581e-06 1.160123e-05 0.0001698510
```

(In the output shown in the box above a large number of output lines have been suppressed and some intermediate columns have been deleted, because the page width does not allow them to be shown completely.)

The first `R` command in the box above calls the function `PSPMequi` with the option argument `c("single")`. This instructs the `PSPMequi` function to compute the equilibrium state for only a single parameter value (in this case the value $3.0 \cdot 10^{-4}$ for the parameter with index 1, which is the parameter R_{max} in the PNAS model; see code block 4.3.2.2). This call to the function `PSPMequi` produces a result file called `PNAS2002-EQ-0000.csb` containing a single set of data for the environmental state variables and the population, which is imported into the `R` workspace using the function `csbread`. This initial state, contained in the variable `initstate`, is subsequently used as initial condition for the numerical integration of the dynamics using the function `PSPMecodyn`.

This starting point of the numerical integration is passed to the function `PSPMecodyn` as its second argument, as shown above, whereas the first argument defines the basename of the file with the model implementation `"PNAS2002"`. The third argument to the function `PSPMecodyn` sets the interval during which a new cohort is formed equal to 1.0, the time interval between output written to the file with extension `.out` also equal to 1.0 and the time interval between complete state output written to the file with extension `.csb` equal to 10.0. As last element of the third argument the maximum integration is set equal to 1000.0. The final argument to the function `PSPMecodyn`, the option argument `c("report", "50")`, forces the function `PSPMecodyn` to only write output to the console every 50 time units. Every 50 time units the function `PSPMecodyn` hence reports the current values of the time and the environmental state variables to the console.

In the `R` command box above the values of the computed points along the trajectory of the

model dynamics are saved in the output list element `output1$curvepoints`, the contents of which are inspected after the `PSPMecodyn` function finishes and it has printed out the textual information about the computation. The output list element `output1$curvepoints` contains in consecutive columns the time value, the values of all environmental state variables, the current population birth rate of all structured populations and the values of all interaction variables defined in the function `Impact()` (see section 4.3.2.11). The demo script `"deRoosPerssonDynamics"` uses the data contained in `output1$curvepoints` to plot the juvenile biomass (the sum of column 7 and 8 of `output1$curvepoints`) as well as the adult biomass (column 9 of `output1$curvepoints`) as a function of time.

In the following `R` command box, another trajectory of the dynamics is computed, but now starting from an initial state generated by the function `PSPMind`. This function computes the individual life history at a specific set of values for the environmental state variables. The details abut the function `PSPMind` are discussed in chapter 8 and hence will not be covered here. The only thing to point out here is that the second argument in the call to the function `PSPMind` is a vector `c(1.561276e-04, 1.270327e-04, 4.008016e-06, 0.01)` with as its first 3 elements the values of the environemental state variables at which to compute the individual life history (see section 4.3.2.1 for the interpretation of these environmental state variables). The last element of this vector (0.01) defines the population birth rate, which value will be used to scale the number of individuals in all cohorts of the population. Larger values of this birth rate will hence imply that the size-structured consumer population is initially larger. The output of the function `PSPMind` is a list with the same structure as produced by the function `PSPMequi` in its state output file with extension `.csb`. This state produced as output by the function `PSPMind` is assigned to the variable `initstate`, which is subsequently used as starting point for the numerical integration. The subsequent call to the function `PSPMecodyn` is identical to the one shown in `R` command box 5.2.A, while also its output is similar to the output shown in command box 5.2.A and discussed above. These will hence not be further discussed here. The demo script `"deRoosPerssonDynamics"` uses also the data contained in `output2$curvepoints` to plot the juvenile biomass (the sum of column 7 and 8 of `output2$curvepoints`) as well as the adult biomass (column 9 of `output2$curvepoints`) of this trajectory as a function of time.

Command box 5.2.B

```
> initstate <- PSPMind("PNAS2002", c(1.561276e-04, 1.270327e-04, 4.008016e-06, 0.01), options = c("isort", "1"))

<...compilation output lines suppressed in this box...>

                                    Istate[ 0]     Istate[ 1]       Survival             R0        Impact[ 0]
                                                  Impact[ 1]        Impact[ 2]        Impact[ 3]
Pop. # 0 - Bstate  0 - (Final):        1248.79        273.555          1E-09              1         0.0521033
             0.014515         0.0426407          0.628847


> output2 <- PSPMecodyn("PNAS2002", initstate, c(1, 1, 10, 1000), options = c("report", "50"))
Dynamic library file PNAS2002ecodyn.so is up-to-date

        0.00, 1.56127600E-04, 1.27032700E-04, 1.00801215E-04
       50.00, 1.32044964E-05, 1.68821129E-04, 2.82764765E-06
      100.00, 2.10361075E-05, 1.52692631E-04, 3.64796500E-06
<...output lines suppressed in this box...>
      900.00, 1.90701042E-04, 1.25787901E-04, 3.13606041E-06
      950.00, 1.96875492E-04, 1.13526555E-04, 3.29816659E-06
     1000.00, 1.81028462E-04, 1.07370790E-04, 3.90578812E-06


RUN PNAS2002-ECODYN-0001 COMPLETED at T = 1000.00:
** Program terminated. Normal closure of output files succeeded.          **

#
# Executing : PSPMecodyn("PNAS2002", <Initial state>, c(1, 1, 10, 1000), NULL, NULL, c("report", "50"))
```

```
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  0.0003         Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
#   Cohort cycle time interval                                          : 1.0
#   Output time interval                                                : 1.0
#   Complete state output interval                                      : 10.0
#   Maximum integration time                                            : 1000.0
#
#
#     1:Time       2:E[0]       3:E[1]       4:E[2]       5:b[0] ..    7:I[0][1]    8:I[0][2]    9:I[0][3]
> output2$curvepoints
          V1           V2           V3           V4           V5 ..           V7           V8           V9
   [1,]    0 1.561276e-04 0.0001270327 1.008012e-04 0.0000000000 .. 1.008012e-04 1.769969e-04 0.0010387677
   [2,]    1 2.133163e-05 0.0001569627 8.017346e-05 0.0011930961 .. 8.017346e-05 1.817787e-04 0.0010364129
   [3,]    2 4.103512e-06 0.0001795904 4.068710e-05 0.0004298024 .. 4.068710e-05 1.806636e-04 0.0010172865
<...output lines suppressed in this box...>
 [999,]  998 0.0001821318 0.0001074357 3.873354e-06 0.0001903570 .. 3.873354e-06 1.988401e-05 0.0001212593
[1000,]  999 0.0001815848 0.0001074010 3.889513e-06 0.0001911060 .. 3.889513e-06 2.009461e-05 0.0001214775
[1001,] 1000 0.0001810285 0.0001073708 3.905788e-06 0.0001918771 .. 3.905788e-06 2.030538e-05 0.0001217083
```

(In the output shown in the box above a large number of output lines have been suppressed and some intermediate columns have been deleted, because the page width does not allow them to be shown completely.)

In the following R command box, another trajectory of the dynamics is computed, but now starting from an initial state that is constructed using list construction commands from the R command line. The first command in the box below constructs a list with elements `Environment` and `Pop00`. The first element `Environment` contains the initial values for the 3 environmental state variables in the PNAS model (see section 4.3.2.1 for the interpretation of these environmental state variables). The last element `Pop00` is a matrix with cohort data for the initial population to start the numerical integration with. This matrix should have 3 columns, specifying the number of individuals in the cohort, their age and their body length, given that in the PNAS model age and length are the two individual state variables. If more columns are specified in the element `Pop00`, they will be ignored. Each row of the matrix contained in `Pop00` specifies one cohort. The first command below shows that the initial population for the following integration consists of a cohort of newborn individuals with age 0 and length $\ell = \ell_b = 7.0$. The density of these newborn individuals is 0.001. In addition, the initial population includes a cohort of adult individuals with age 300 and length $\ell = 111$ with a cohort density equal to $1.0 \cdot 10^{-5}$. The list `initstate` that is thus produced in the first command in the box below is subsequently used as starting point for the numerical integration. The subsequent call to the function `PSPMecodyn` is identical to the one shown in R command box 5.2.A, while also its output is similar to the output shown in command box 5.2.A and discussed above. These will hence not be further discussed here. The demo script `"deRoosPerssonDynamics"` uses also the data contained in `output3$curvepoints` to plot the juvenile biomass (the sum of column 7 and 8 of `output3$curvepoints`) as well as the adult biomass (column 9 of `output3$curvepoints`) of this trajectory as a function of time.

Command box 5.2.C

```
> initstate <- list(Environment = c(1.561276e-04, 1.270327e-04, 4.008016e-06), Pop00 = matrix(c(0.001, 0, 7.0,
    1.0E-5, 300, 111), ncol = 3, byrow = TRUE))
> output3 <- PSPMecodyn("PNAS2002", initstate, c(1, 1, 10, 1000), options = c("report", "50"))
Dynamic library file PNAS2002ecodyn.so is up-to-date

       0.00, 1.56127600E-04, 1.27032700E-04, 3.08700000E-06
```

```
       50.00, 1.27379538E-04, 1.39392354E-04, 4.48274388E-06
      100.00, 1.34147338E-04, 1.42360458E-04, 3.99924658E-06
<...output lines suppressed in this box...>
      900.00, 1.29856560E-04, 1.27516287E-04, 4.56714543E-06
      950.00, 1.27055262E-04, 1.35483257E-04, 4.37381850E-06
     1000.00, 1.36470065E-04, 1.39125331E-04, 4.05246328E-06


RUN PNAS2002-ECODYN-0002 COMPLETED at T = 1000.00:
** Program terminated. Normal closure of output files succeeded.          **

#
# Executing : PSPMecodyn("PNAS2002", <Initial state>, c(1, 1, 10, 1000), NULL, NULL, c("report", "50"))
#
# Parameter values  :
#
#   Rho       :  0.1            Rmax      :  0.0003         Lb        :  7
#   Lv        :  27             Lj        :  110            Lm        :  300
#   Beta      :  9E-06          Imax      :  0.0001         Rh        :  1.5E-05
#   Gamma     :  0.006          Rm        :  0.003          Mub       :  0.01
#   A         :  5000           Th        :  0.1            Epsilon   :  0.5
#   Delta     :  0.01
#
#   Cohort cycle time interval                                          : 1.0
#   Output time interval                                                : 1.0
#   Complete state output interval                                      : 10.0
#   Maximum integration time                                            : 1000.0
#
#
#     1:Time       2:E[0]       3:E[1]       4:E[2]       5:b[0] ..    7:I[0][1]    8:I[0][2]    9:I[0][3]
> output3$curvepoints
          V1           V2           V3           V4           V5 ..           V7           V8           V9
   [1,]    0 0.0001561276 0.0001270327 3.087000e-06 0.0000000000 .. 3.087000e-06            0 0.0001230868
   [2,]    1 0.0001545920 0.0001269028 4.032747e-06 0.0003394561 .. 4.032747e-06            0 0.0001250936
   [3,]    2 0.0001529027 0.0001270109 4.607412e-06 0.0003415640 .. 4.607412e-06            0 0.0001270795
<...output lines suppressed in this box...>
 [999,]  998 0.0001359284 0.0001390871 4.065588e-06 0.0003399111 .. 4.065588e-06 7.546175e-06 0.0002128082
[1000,]  999 0.0001361979 0.0001391073 4.059020e-06 0.0003393768 .. 4.059020e-06 7.490502e-06 0.0002126396
[1001,] 1000 0.0001364701 0.0001391253 4.052463e-06 0.0003388324 .. 4.052463e-06 7.436078e-06 0.0002124645
```

(In the output shown in the box above a large number of output lines have been suppressed and some intermediate columns have been deleted, because the page width does not allow them to be shown completely.)

5.3 Output files generated by the PSPMecodyn function

The computational module that is produced by the PSPMecodyn function generates 3 output files. The name of these files is always of the form <Modelname>-ECODYN-<NNNN>.<ext>, in which <Modelname> is the same as the name of the file specifying the model excluding its '.R' or '.h' extension, <NNNN> is a 4-digit number that is unique for the current computation and .<ext> is the extension, which can be either .csb, .err or .out. The unique number distinguishes the same types of curve computations for the same model from each other. The number is obtained by considering increasing values of <NNNN> (i.e., 0000, 0001, 0002 and so forth) and testing whether result files with the particular index are already present. The program uses the first value of <NNNN> that is not in use.

The file called <Modelname>-ECODYN-<NNNN>.err that is generated during the computations contains information about the numerical progress of the computations. It reports details on the steps taken during the numerical integration, such as step sizes used, number of successful and failed integration steps and information about the detection of stage transitions. The amount of detail is dependent on the value of the option "report". The default behavior is to produce no output at all (c("report",

"0"); see section 5.1). This file can be informative in case the computation of a particular curve stops for unknown reasons, but is otherwise of little use.

The output file called <Modelname>-ECODYN-<NNNN>.out holds the same information as is contained in the two elements of the output list returned by the PSPMecodyn function, output$curvepoints and output$curvedesc (see the help page on PSPMecodyn using ?PSPMecodyn). The first lines of this file all start with a '#' sign and contain the information about the run performed, which is also contained in output$curvedesc and can be listed by the statement cat(output$curvedesc, sep='\n'). Following this descriptive header the file contains columns with computational results that are also contained in the variable output$curvepoints (see, for example, R command box 5.2.A). In fact, the two elements of the output list, output$curvepoints and output$curvedesc, are generated by reading the contents of the file <Modelname>-ECODYN-<NNNN>.out from disk after the computations have ended, storing all lines that start with a '#' sign into a single string variable output$curvedesc, while storing the information on all other lines into the data matrix output$curvepoints.

The file called <Modelname>-ECODYN-<NNNN>.csb is an output file containing the values of all environment variables and the distribution of all structured populations in the model. This is a binary file, the content of which can be accessed from R using the function csbread. Output is written to this output file at regular time intervals, where the interval between consecutive output times is specified by the third element of the obligatory argument timepars to the function PSPMecodyn (see section 5.1). For example, the file PNAS2002-ECODYN-0002.csb is generated by the invocation of the PSPMecodyn function in R command box 5.2.C. Its contents can be listed by:

Command box 5.3.A

```
> csbread("PNAS2002-ECODYN-0002.csb")

States in file PNAS2002-ECODYN-0002.csb:

    1: State-0.000000E+00
    2: State-1.000000E+01
    3: State-2.000000E+01
<...output lines suppressed in this box...>
   99: State-9.800000E+02
  100: State-9.900000E+02
  101: State-1.000000E+03
```

The structure called State-2.000000E+01 contains the population state at time point $t = 20.0$ during the simulation of the ecological dynamics, as its name suggests. Its contents can be read into the workspace by issuing the command csbread("PNAS2002-ECODYN-0002.csb",3) or csbread("PNAS2002-ECODYN-0002.csb","State-2.000000E+01").

Loading this state into the R workspace reveals it to be a list containing various arrays of numbers, as shown in the following box:

Command box 5.3.B

```
> popstate <- csbread("PNAS2002-ECODYN-0002.csb","State-2.000000E+01")
> popstate
$Time
[1] 20

$Parameters
 [1] 1.0e-01 3.0e-04 7.0e+00 2.7e+01 1.1e+02 3.0e+02 9.0e-06 1.0e-04 1.5e-05 6.0e-03 3.0e-03 1.0e-02 5.0e+03
     1.0e-01 5.0e-01 1.0e-02

$Environment
[1] 1.364491e-04 1.324733e-04 4.008016e-06

$Pop00
          Density    Istate00   Istate01     Istate02 Istate03 Istate04 Istate05 Istate06 Istate07
```

```
 [1,] 2.687298e-04   0.4443476   7.700723 0.0002687298        0        7        0        0       19
 [2,] 1.371993e-04   1.4444489   9.272342 0.0002681358        0        7        0        0       18
 [3,] 7.013400e-05   2.4445505  10.835272 0.0002674988        0        7        0        0       17
<...output lines suppressed in this box...>
[20,] 6.246577e-08  19.4462036  36.144616 0.0002496189        0        7        1        0        0
[21,] 1.892823e-07  20.0000000  36.932027 0.0010000000        0        7        1        0        0
[22,] 8.187308e-06 320.0000000 129.171752 0.0010000000        0        7        2        0        0
```

The first element `$Time` in the list `popstate` contains the value of the integration time t at which the population state is stored. The second element of the list, a vector called `$Parameters`, contains the values of all the model parameters used in the numerical integration. The third element of the list is a vector called `$Environment` containing the values of the environmental state variables at time t. The last element in the list called `$Pop00` is a two-dimensional array characterising the population distribution at time t with the first column `$Pop00[,1]` representing the number of individuals in a particular cohort in the population and the subsequent columns `$Pop00[,2]` and `$Pop00[,3]` representing the average values of their individual state variables with index 0 and 1 (corresponding to individual age and body size in the PNAS model, see section 4.2), as shown in the `R` command box above. If individuals would be characterized by more than two individual state variables, the values of these would follow in additional columns of the two-dimensional array `$Pop00`. The additional columns with labels `Istate02` to `Istate07` are used by the `PSPMecodyn` function internally for bookkeeping purposes. The first column labeled `Istate02` represents the initial number of individuals in the cohort at the moment it was formed. The following two columns, labeled `Istate03` and `Istate04`, specify the state at birth of the individuals in the cohort, which in case of the PNAS model equals age 0 and length $\ell = \ell_b = 7.0$, respectively. In case the number of individual state variables is larger, the number of these columns representing the state at birth will increase accordingly. The next column, labeled `Istate05`, indicates the life stage that the individuals in the cohort are currently in. In the PNAS model life stage 0, 1 and 2 refers to the juveniles that are vulnerable to predation, juveniles that are invulnerable to predation and adult individuals, respectively. The next column, labeled `Istate06`, contains the index of the state at birth, with which individuals are born. Finally, the last column, labeled `Istate07`, contains the time value at which the cohort was formed.

The `R` command box above also illustrates that the dimension of the array `$Pop00` indicates that the population at time t consists of 22 cohorts. However, this number of cohorts will vary over time and is here still rather low because the initial population consisted of only two cohorts of individuals (see command box 5.2.C).

Chapter 6

Analysis of evolutionary fixed points of non-linear PSPMs

6.1 Theoretical and computational background

The analysis of evolutionary fixed points of non-linear PSPMs focuses on the question how the value of a particular model parameter would change if mutations would generate variability in this parameter value and selection would act on this variability. Adaptive dynamics (Metz et al. 1996, Geritz et al. (1998)) constitutes an approach to answer such questions, while carefully taking into account the feedback of populations on their environment. The central function in the theory of adaptive dynamics is the long-term population growth of a mutant type in an environment that is completely dominated and hence determined by a resident population. This quantity is usually referred to with the symbol $s_x(y)$, in which x refers to the type of the resident population and y refers to the type of the mutant. Not surprisingly, when the mutant is identical to the resident it has a population growth rate 0, since the resident is assumed to persist indefinitely (Notice that this does no require the population to be in equilibrium). Therefore:

$$s_x(y)|_{y=x} = 0$$

Furthermore, the partial derivative $\partial s_x(y)/\partial y$ equals the *selection gradient*, indicating whether a mutation-selection process will lead to larger or smaller values of the trait x. If

$$\left.\frac{\partial s_x(y)}{\partial y}\right|_{y=x} > 0 \tag{6.1}$$

a mutant with a trait value y larger than the resident trait value x will have a positive long-term growth rate and hence will be able to invade, while the opposite holds for when the partial derivative is negative. An evolutionary singular point, which will be indicated with x^*, now occurs where

$$\left.\frac{\partial s_x(y)}{\partial y}\right|_{y=x^*} = 0$$

Furthermore, The second-order partial derivatives

$$\left.\frac{\partial^2 s_x(y)}{\partial x^2}\right|_{y=x^*} \quad \text{and} \quad \left.\frac{\partial^2 s_x(y)}{\partial y^2}\right|_{y=x^*}$$

determine whether the evolutionary singular point is a convergent stable strategy (CSS), an evolutionary repellor (ERP) or an evolutionary branching point (EBP) (Geritz et al. 1998).

In the bifurcation analysis of PSPMs the equilibrium of a structured population is determined by the condition

$$R_0 - 1 = 0$$

in which R_0 is the expected number of offspring produced by a single individual of the structured population during its entire life. R_0 is not the same as the long-term population growth rate, but the condition $R_0 - 1$ is sign-equivalent with the population growth rate: the sign of $R_0 - 1$ and the population growth rate are always the same and when $R_0 - 1$ equals 0, the population growth rate is 0 as well. According to the theory of adaptive dynamics (Metz et al. 1996, Geritz et al. (1998)) the function $R_0 - 1$ can therefore be used for the analysis of evolutionary fixed points of PSPMs.

In the context of the PSPMs the traits x and y will refer to the resident and mutant value, respectively, of one of the model parameters. The value of such a parameter will influence the expected number of offspring produced by a single individual of the structured population during its entire life, R_0, if the parameter represents a life history characteristic. On the other hand, R_0 is also influenced by the environment in which the individual lives. A key element of PSPMs is that this environment itself is influenced by the structured population to such an extent that the equilibrium value of the environment is determined by the population. The equilibrium value of the environment is hence also a function of the model parameters and we can write the equilibrium condition of the structured population more appropriately as:

$$R_0(y, \tilde{E}(x))|_{y=x} - 1 = 0$$

In this condition x refers to the value of one of the model parameters in the PSPM of the resident type of individual that dominates the structured population and hence determines the equilibrium value of the environment variables $\tilde{E}(x)$, whereas y refers to the value of that same parameter for a mutant type, which invades the population at low density. The partial derivatives of the function $R_0(y, \tilde{E}(x)) - 1$ can therefore be used to classify a computed equilibrium in a PSPM as an evolutionary fixed point and determine whether it is a convergent stable strategy (CSS), an evolutionary repellor (ERP) or an evolutionary branching point (EBP) (Geritz et al. 1998). Since the constant 1 in this function is irrelevant for the partial derivatives, the quantities of interest are:

$$R_{0x} := \left.\frac{\partial R_0(y, \tilde{E}(x))}{\partial y}\right|_{y=x^*}$$

$$R_{0xx} := \left.\frac{\partial^2 R_0(y, \tilde{E}(x))}{\partial x^2}\right|_{y=x^*}$$

$$R_{0yy} := \left.\frac{\partial^2 R_0(y, \tilde{E}(x))}{\partial y^2}\right|_{y=x^*}$$

During an equilibrium computation with the `PSPMequi` script the program can check for every computed equilibrium point the value of R_{0x}. This test is, however, only performed when the option `"popEVO"` is set to a valid value, that is in the range 0 to `POPULATION_NR-1`. The value of this option identifies the index of the structured population, for which to carry out the evolutionary fixed point analysis. As default the option `"popEVO"` is not defined and the test of the evolutionary properties of the equilibrium is skipped, as was the case in the model analyzed in section 4.4. When the software detects a sign change in this quantity, it attempts to locate the exact position of the evolutionary fixed point by solving for the equilibrium of the PSPM with the additional condition $R_{0x} = 0$. When

successful the software computes the second-order partial derivatives R_{0xx} and R_{0yy} to classify the evolutionary fixed point. The computation of these partial derivatives is done entirely numerically using a central-differencing approach. Unless it fails to compute one of the partial derivatives properly, the software will report whether a convergent stable strategy (CSS), an evolutionary repellor (ERP) or an evolutionary branching point (EBP) has been detected.

Once an evolutionary fixed point is detected, the software allows for 2 further steps of analysis of the evolutionary fixed point. The first type of analysis that can be carried out is that the evolutionary fixed point can be computed for a range of values of a second model parameter. More precisely, the condition $R_{0x} = 0$ is added as supplementary condition to the system of equations determining the equilibrium of the PSPM and because of this additional condition one more unknown quantity, the value of a second model parameter, has to be solved for. This type of computations is referred to with the acronym `"ESS"`. They yield curves that show the evolutionary stable value of the evolutionary parameter as a function of the first bifurcation parameter.

The software is in fact sufficiently general to allow for continuation of curves with multiple model parameters having their evolutionary stationary value. These curves are all indicated with the acronym `"ESS"`. For each evolutionary parameter the corresponding condition $R_{0x} = 0$ is added to the system of equations to solve. For each parameter at its evolutionary stationary value the vector of initial estimates of a solution point (the third argument to the function `PSPMequi`) should contain a value close this evolutionary stationary value, whereas the index of the parameter in the model and its allowable minimum and maximum value are defined by the triplet in the fifth argument to the function `PSPMequi` (see section 4.4.2). As discussed in section 4.4.2 this fifth argument to the function `PSPMequi` should for `"ESS"` computations contain at least 2 triplets, one for the (first) bifurcation parameter and one for the model parameter that is fixed at its evolutionary stationary value, but it can be extended with more triplets in case the `"ESS"` curve is characterized by multiple parameters at their evolutionary stationary value. The number of triplets for evolutionary parameters should match the number of initial estimates for these parameters in the third argument to the function `PSPMequi`. The current version of the software computes for each parameter at its evolutionary stationary value the second-order partial derivatives R_{0xx} and R_{0yy} and writes these second-order derivatives to the output file. Notice, however, that these derivatives only provide a classification of the evolutionary stationary point in terms of convergent stable, evolutionary repellor or evolutionary branching point in the case of a single evolutionary parameter, as the classification of multidimensional evolutionary fixed point involves more complex computations of derivatives (see Leimar (2005)).

The second type of analysis that can be performed is the computation of the pairwise invasibility plot (or PIP; for an explanation see Geritz et al. (1998)) starting from the evolutionary fixed point. This type of computation is indicated with the acronym `"PIP"` and is carried out by supplementing the system of equations determining the equilibrium of the PSPM with the condition $R_0(y, \tilde{E}(x)) = 1$. Because of this extension, one more unknown variable has to be solved for, which in this case is the mutant value of the model parameter y. The first and second bifurcation parameter in this case have the same index in the array of parameter values, but the first bifurcation parameter refers to the resident value x of this parameter, while the second bifurcation parameter refers to the mutant value y. The result of such a computation is a curve in the parameter space spanned by x and y, where the growth rate of a mutant with parameter value y in an equilibrium environment determined by a resident population with parameter value x has a zero population growth rate. PIPs are plots of such curves and these plots can be used for inferring various evolutionary consequences (Geritz et al. 1998).

While performing computations of the type `"ESS"` and `"PIP"` the software continuously computes the value of the second order partial derivatives R_{0xx} and R_{0yy} and writes these values to the output file (with extension `.out`). Inspection of the output file can hence also indicate whether an evolutionary fixed point changes its type, for example from CSS to EBP or vice versa. Automatic detection and processing of such type changes is, however, (currently) not implemented in the software.

6.2 An example model for the analysis of evolutionary fixed points

The analysis of evolutionary fixed points of PSPMs will be illustrated using a model for a size-structured consumer population feeding on a resource R. Individual consumers are assumed to be born at size s_b and forage on the resource at a rate proportional to an allometric function of their size, s^q. They are furthermore assumed to have a linear functional response, such that their ingestion rate equals $\gamma(s, R) = I_{max} R s^q$. Ingested energy is assimilated with a constant conversion efficiency σ. Maintenance costs are also assumed to follow an allometric relation of body size, $T s^p$.

Juvenile individuals spend all their net energy production on growth in body size and hence have a somatic growth rate $\sigma\gamma(s, R) - T s^p$. Above a body size threshold $s = s_j$, referred to as the maturation size, individuals decrease the fraction of their net-energy production that they invest in somatic growth and use the remainder for investments in reproduction. The function $\kappa(s)$ indicates the fraction of net-energy production invested in somatic growth. $\kappa(s)$ is a cubic function of size that decreases smoothly and continuously from a value of 1 at $s = s_j$ to a value 0 at $s = s_m$. The size threshold $s = s_m$ hence represents the maximum body size individual consumers can possibly reach. The energy invested into reproduction is converted into offspring with size $s = s_b$. No further conversion losses are assumed to occur during somatic growth and reproduction, the conversion efficiency σ is assumed to include all such losses. The somatic growth rate hence equals $g(s, R) = \kappa(s)(\sigma\gamma(s, R) - T s^p)$, while the fecundity is given by $\beta(s, R) = (1 - \kappa(s))(\sigma\gamma(s, R) - T s^p)/s_b$. Consumers experience a constant, size-independent mortality. Finally, in the absence of consumers the resource follows semi-chemostat dynamics with turn-over rate δ and maximum resource density R_{max}.

The model dynamics can now be described by the following system of partial and ordinary differential equations for the resource density R and the consumer size distribution $n(t, s)$:

$$\frac{\partial n(t,s)}{\partial t} + \frac{\partial \left(g(s,R) n(t,s)\right)}{\partial s} = -\mu\, n(t,s)$$

$$g(s_b, R)\, n(t, s_b) = \int_{s_b}^{s_m} \beta(s, R)\, n(t, s)\, ds$$

$$\frac{dR}{dt} = \delta\,(R_{max} - R) - \int_{s_b}^{s_m} \gamma(s, R)\, n(t, s)\, ds$$

As discussed above the individual life history functions representing food ingestion, somatic growth, fecundity and the fraction of net-energy production allocated to somatic growth are given by:

$$\gamma(s, R) = I_{max}\, R\, s^q$$

$$g(s, R) = \kappa(s)\,(\sigma\gamma(s, R) - T\, s^p)$$

$$\beta(s, R) = \frac{(1 - \kappa(s))\,(\sigma\gamma(s, R) - T\, s^p)}{s_b}$$

$$\kappa(s) = \begin{cases} 1 & \text{if } s \le s_j \\ 1 - 3\left(\dfrac{s - s_j}{s_m - s_j}\right)^2 + 2\left(\dfrac{s - s_j}{s_m - s_j}\right)^3 & \text{otherwise} \end{cases}$$

The evolutionary fixed point analysis will focus on the parameter q, the allometric scaling exponent of ingestion rate with body size s. Default values of the other parameters are: $\delta = 0.1$, $R_{max} = 2.0$, $I_{max} = 1.0$, $T = 0.1$, $p = 1.0$, $s_b = 0.05$, $s_j = 1.0$, $s_m = 2.0$ and $\sigma = 0.5$. The background mortality experienced by consumers is assumed to equal $\mu = 0.01$.

The model is implemented in the model-specific file `Indet_growth.h`, which can be opened by executing the command `showpspm("Indet_growth.h")`. The implementation of the model follows the guidelines as presented for the PNAS model in section 4.2 and will therefore not be discussed in detail. The reader is encouraged to inspect the file `Indet_growth.h` and work out the translation of the mathematical formulation given above into the necessary `C`-code elements required for analysis.

6.3 Model analysis

The analysis can be performed by executing the demo `"Indet_growth"`, which is included with this `R` package (execute it using `demo("Indet_growth", echo = FALSE)`). Below 3 commands will be discussed that are executed by the demo script `"Indet_growth"` and that illustrate the possibilities to use the software for evolutionary fixed point analysis. The demo script `"Indet_growth"` furthermore performs some plotting of the output data generated by the computational module.

The analysis starts out by computing the equilibrium of the consumer-resource model as a function of the parameter q, the allometric scaling exponent of ingestion with body size s. This parameter has index 6 in the parameter array defined in the model-specific file `Indet_growth.h` (see line 50–80 in that file). The computation starts from an equilibrium point at $q = 1.0$, computing the equilibrium for decreasing values of q in the range $0.5 \leq q \leq 2.0$. Hence, the fourth and fifth argument of the `PSPMequi` function, which specify the step size along the equilibrium curve and the index of the bifurcation parameter plus the limits to its range for the computation, respectively, are taken equal to `-0.1` and `c(6,0.5,2.0)`.

The initial point for the computations is a rather crude estimate of the equilibrium state for $q = 1$, for which parameter value all rates are linear in body mass s. Per unit biomass the net-production rate of new biomass, either through somatic growth or through fecundity, then equals $\sigma I_{max} R - T$, while the loss rate per unit biomass equals the mortality rate μ. Equating these two rates to each other, yields the equilibrium resource density $\tilde{R} = (T + \mu)/(\sigma I_{max}) = 0.22$. The initial estimate for the population birth rate in equilibrium is especially crude, as it is taken equal to 0. Despite that the initial point is not close to the equilibrium solution for $q = 1$ the computations easily converge as can be seen in the `R` command box below.

Furthermore, the computations are carried out with the default parameter values, such that the 6th argument of the function `PSPMequi` is left undefined (`NULL`). The last argument of the function `PSPMequi` is the option vector, in which the option `"popEVO"` is set equal to `"0"`. Defining this option instructs the program to compute the selection gradient in the evolutionary parameter during equilibrium continuations (Curve type `"EQ"`). The evolutionary parameter can be any of the parameters in the parameter vector by assigning the index of the particular parameter to the option `"parEVO"` in the option vector. However, only when the option `"parEVO"` is set equal to the bifurcation parameter or left unspecified (in which case `parEVO` defaults to the bifurcation parameter) is it possible to assess whether or not a computed equilibrium is an evolutionary fixed point. Below the option `"popEVO"` identifies the structured population with index `0` as the population for which to carry out the evolutionary analysis (obviously, as it is the only population in this problem).

Command box 6.3.A

```
> output1 <- PSPMequi("Indet_growth","EQ",c(1.0,0.22,0.0),-0.1,c(6,0.5,2.0),NULL,c("popEVO","0"))

Building executable Indet_growthequi.so ...

<...compilation output lines suppressed in this box...>

  1.00000000E+00  2.20000000E-01  3.55373787E-02
  9.98043762E-01  2.19653632E-01  3.45381701E-02
  9.96005285E-01  2.19313094E-01  3.35389579E-02
```

```
<...output lines suppressed in this box...>
  9.44570226E-01  2.15621677E-01  1.85536239E-02
  9.38263361E-01  2.15592978E-01  1.75885671E-02  ****  CSS #0  ****
  9.38034889E-01  2.15593014E-01  1.75557539E-02
<...output lines suppressed in this box...>
  5.18672354E-01  2.40531837E-01  7.04499110E-03
  5.09283848E-01  2.41208401E-01  7.01122421E-03
  4.99855837E-01  2.41890186E-01  6.97858797E-03

#
# Executing : PSPMequi("Indet_growth", "EQ", c(1, 0.22, 0), -0.1, c(6, 0.5, 2), NULL, c("popEVO", "0"))
#
# Parameter values  :
#
#   Delta     :  0.1            Rmax      :  2              Sb        :  0.05
#   Sj        :  1              Sm        :  2              Imax      :  1
#   q         :  1              Sigma     :  0.5            T         :  0.1
#   p         :  1              Mu        :  0.01
#
# Index of bifurcation parameter #1                          : 6
#
#           1:q    2:E[0]     3:b[0] 4:I[0][0]  5:I[0][1] 6:I[0][2]   7:R0[0]     8:R0_x[6]   9:RHS norm
#

> output1$curvepoints
             V1        V2         V3        V4         V5        V6        V7             V8           V9
 [1,] 1.0000000 0.2200000 0.03553738 0.1780000 0.53230236 0.2767885 1.0000000 -4.144555e+01 9.579106e-09
 [2,] 0.9980438 0.2196536 0.03453817 0.1780346 0.52939068 0.2798577 1.0000000 -3.894615e+01 3.294274e-08
 [3,] 0.9960053 0.2193131 0.03353896 0.1780687 0.52628792 0.2831152 1.0000000 -3.648053e+01 3.052791e-08
<...output lines suppressed in this box...>
[18,] 0.9445702 0.2156217 0.01855362 0.1784378 0.43178150 0.3792995 1.0000000 -2.091709e+00 1.618872e-08
[19,] 0.9382634 0.2155930 0.01758857 0.1784407 0.41943590 0.3916582 1.0000000 -9.063882e-11 7.983139e-09
[20,] 0.9380349 0.2155930 0.01755575 0.1784407 0.41899094 0.3921031 1.0000000  7.059485e-02 8.044140e-09
<...output lines suppressed in this box...>
[56,] 0.5186724 0.2405318 0.00704499 0.1759468 0.10149399 0.6982643 1.0000000  6.421252e+00 1.465988e-09
[57,] 0.5092838 0.2412084 0.00701122 0.1758792 0.09943055 0.7000202 1.0000000  6.322922e+00 2.255060e-09
[58,] 0.4998558 0.2418902 0.00697859 0.1758110 0.09742641 0.7017144 1.0000000  6.227017e+00 1.991225e-09

> output1$bifpoints
             V1        V2         V3        V4         V5        V6        V7             V8           V9
[1,]  0.9382634  0.215593 0.01758857 0.1784407  0.4194359 0.3916582         1 -9.063882e-11 7.983139e-09

> output1$biftypes
[1] "CSS #0"
```

As can be seen in the command box above an evolutionary fixed point is detected at $q^* = 0.938266$. On the basis of the second-order partial derivatives, which are not reported explicitly by the program during this computation, the fixed point is classified as a convergent stable strategy. It occurs (self-evidently in this model) in the structured population with index 0. The output element `output1$biftypes` therefore consists of the single string `"CSS #0"`.

Because the program is instructed to assess the evolutionary properties of the computed equilibrium points, the output matrix `output1$curvepoints` produced by the call to `PSPMequi`, has an additional column of output compared to the columns of output discussed in section 4.4, labeled `R0_x[0]` (column 8). This is the derivative of the R_0 value, the expected number of offspring produced by an individual during its entire life, with respect to the bifurcation parameter q for the population with index 0. Inspection of this column shows that this derivative is negative for q-values larger than q^* and positive for q-values smaller than this evolutionary fixed point value. This implies that for $q > q^*$ a mutation-selection process will select for smaller values of the trait q, while larger q-values will be selected for when $q < q^*$. The evolutionary fixed point q^* is therefore convergent. The demo script `"Indet_growth"` illustrates this computation by plotting the equilibrium resource density and the equilibrium consumer biomass as a function of the parameter q. At the CSS the equilibrium resource

density reaches a minimum value.

In the next step of the analysis the detected evolutionary fixed point is used as starting point for a computation of its value as a function of a second model parameter, p, the allometric scaling exponent of the maintenance rate with body size. The acronym of this type of computation is `"ESS"`, which is supplied as the second argument to the `PSPMequi` function in the next `R` command box. Such `"ESS"` computations always use a parameter that is not defined to have its evolutionary stationary value as (first) bifurcation parameter. The evolutionary parameters are added as additional variables to the problem, which are following all values for the environmental variable(s) and population birth rate(s). Since the default parameter value for p is 1.0, the initial point of the `"ESS"` computation below is specified as `c(1.0,output1$bifpoints[c(2,3,1)])`, given that the second, third and first element of the array `bdata1` represent the equilibrium resource density, equilibrium birth rate and the evolutionary stationary value q-value in the evolutionary fixed point, respectively.

The fifth argument to the function `PSPMequi`, specifying the indices and range limits of the variable parameters in the computation, now has to contain as first triplet the index, minimum and maximum value of the bifurcation parameter p in the problem, while the second triplet specifies the index, minimum and maximum value of the parameter q, which is assumed to have its evolutionary stationary value. The parameter p has index 9 in the parameter array defined in the model-specific file `Indet_growth.h` (see line 50–80 in that file) and for the computation its value is restricted to the interval $0.5 \leq p \leq 2.0$. The array `c(9,0.5,2.0,6,0.5,2.0)` is therefore passed as 5th argument to the `PSPMequi` function.

Lastly, as in the invocation of the function `PSPMequi` shown in command box 6.3.A the option vector that is supplied to the `PSPMequi` function as 7th argument equals `c("popEVO","0")` to indicate that the evolutionary computations should focus on the R_0 value of the structured population with index 0.

Command box 6.3.B

```
> output2 <- PSPMequi("Indet_growth","ESS",c(1.0,output1$bifpoints[c(2,3,1)]),-0.1,c(9,0.5,2.0,6,0.5,2.0),
                      NULL,c("popEVO","0"))

Dynamic library file Indet_growthequi.so is up-to-date

  1.00000000E+00  2.15592978E-01  1.75885671E-02  9.38263361E-01
  9.52112206E-01  2.15270182E-01  1.68344054E-02  8.93418030E-01
  9.02584076E-01  2.14960684E-01  1.61012822E-02  8.46885465E-01
<...output lines suppressed in this box...>
  5.46566010E-01  2.13317616E-01  1.20009255E-02  5.08772277E-01
  5.39357300E-01  2.13292903E-01  1.19355247E-02  5.01872554E-01
  5.32149192E-01  2.13268473E-01  1.18707128E-02  4.94971646E-01

#
# Executing : PSPMequi("Indet_growth", "ESS", c(1, 0.215593, 0.0175886, 0.938263), -0.1, c(9, 0.5, 2, 6, 0.5,
    2), NULL,
                      c("popEVO", "0"))
#
# Parameter values  :
#
#   Delta     :  0.1            Rmax      :  2              Sb        :  0.05
#   Sj        :  1              Sm        :  2              Imax      :  1
#   q         :  0.938263       Sigma     :  0.5            T         :  0.1
#   p         :  1              Mu        :  0.01
#
# Index of bifurcation parameter #1                           : 9
# Index of parameter #1 at ESS value                          : 6
# Index of structured population for evolutionary analysis    : 0
#
#           1:p    2:E[0]     3:b[0]       4:q  .. 8:R0[0]  9:R0_x[9]  10:R0_xx[0]   11:R0_yy[0]  12:RHS norm
#
```

```
> output2$curvepoints
             V1        V2         V3        V4  ..      V8         V9          V10           V11          V12
 [1,] 1.0000000 0.2155930 0.01758857 0.9382634  ..       1 -1.5738698 0.0001364242 -0.0001362397 1.863626e-07
 [2,] 0.9521122 0.2152702 0.01683441 0.8934180  ..       1 -1.4786767 0.0001259623 -0.0001259056 1.668385e-09
 [3,] 0.9025841 0.2149607 0.01610128 0.8468855  ..       1 -1.3850836 0.0001152072 -0.0001151722 1.916194e-09
<...output lines suppressed in this box...>
[44,] 0.5465660 0.2133176 0.01200093 0.5087723  ..       1 -0.8414241 0.0045940400 -0.0045956900 7.193510e-10
[45,] 0.5393573 0.2132929 0.01193552 0.5018725  ..       1 -0.8324827 0.0044773900 -0.0044790100 6.769479e-10
[46,] 0.5321492 0.2132685 0.01187071 0.4949716  ..       1 -0.8236145 0.0043610400 -0.0043635600 6.244701e-10
```

(Notice that some of the intermediate columns of output have been suppressed in the `R` command box above to keep the displayed output within the page width).

The most important quantities to observe in the output above are the columns in the data matrix `output2$curvepoints` labelled `R0_xx` and `R0_yy`, which represent the second-order partial derivatives of the R_0 value with respect to the resident and mutant value of the parameter q, respectively. As discussed in section 6.1 these second-order partial derivatives classify the evolutionary fixed point as a convergent stable strategy, an evolutionary repellor or an evolutionary branching point (see Geritz et al. (1998) for details). The output shown above indicates that the evolutionary fixed point remains a convergent stable strategy over the entire range of parameters for which the curve is computed, because `R0_xx` is always larger than `R0_yy`.

The graphical illustration produced by the demo script `"Indet_growth"` for this computation consists of the curve of the evolutionary fixed point value of q (column 4 in the data matrix `output2$curvepoints`) as a function of the bifurcation parameter p (column 1 in the data matrix `output2$curvepoints`).

Notice that the quantities `R0_xx` and `R0_yy` are only relevant in the 1-dimensional case, that is if only a single life history is assumed to adopt its evolutionary stationary value in the `ESS` continuation. In the multi-dimensional case, when computing curves of evolutionary stationary points with multiple life history traits evolving, the situation is more complicated. During such multi-dimensional `ESS` continuations, the program does not report the quantities `R0_xx` and `R0_yy`, but instead provides as output the dominant eigenvalues of the Jacobian and Hessian matrices of the canonical equation, as well as the quantity $z^T C_{01} z$, which determines whether or not evolutionary branching can occur at an evolutionary stationary state that is attracting, but not evolutionary stable. In this expression z is the dominant eigenvector of the Hessian matrix and $C_{01} = J - H$, the matrix with cross-derivatives of the canonical equation with respect to the mutant and the resident traits. For more details, see Leimar (2005) and Geritz, Metz, and Rueffler (2016).

The last step in the analysis of the evolutionary fixed point is to construct the pairwise invasibility plot or PIP, starting from the detected evolutionary fixed point. The type of computation is now specified as `"PIP"`. The third argument in the call to the function `PSPMequi`, representing the starting point of the computation, equals `c(output1$bifpoints[c(1,2,3,1)])`, which array contains in addition to the resident value of the parameter q, the equilibrium resource density and the equilibrium population birth rate the mutant value of the parameter q. In the detected evolutionary fixed point this mutant parameter value equals the resident value. Because the resident and mutant parameter are two values of the same model parameter the two triplets that make up the fifth argument to the function `PSPMequi` are identical. This argument hence equals `c(6,0.5,2.0,6,0.5,2.0)`. The function is moreover called twice, once with a positive step size of 0.1 and once with a negative step size of -0.1, to compute the boundary in the PIP that radiates out from the evolutionary fixed point in two directions.

Command box 6.3.C

```
> output3 <- PSPMequi("Indet_growth","PIP",c(output1$bifpoints[c(1,2,3,1)]),0.1,c(6,0.5,2.0,6,0.5,2.0),
                      NULL,c("popEVO","0"))

Dynamic library file Indet_growthequi.so is up-to-date
```

```
5
  9.38263361E-01  2.15592978E-01  1.75885671E-02  9.38263361E-01
  9.44757579E-01  2.15623452E-01  1.85841380E-02  9.31442039E-01
  9.50520866E-01  2.15706603E-01  1.95802172E-02  9.24768482E-01
<...output lines suppressed in this box...>
10  1.05716678E+00  2.40976093E-01  8.16044991E-02  5.12503828E-01
  1.05797641E+00  2.41437457E-01  8.23722323E-02  5.06112695E-01
  1.05878535E+00  2.41902724E-01  8.31367021E-02  4.99682764E-01

#
15 # Executing : PSPMequi("Indet_growth", "PIP", c(0.938263, 0.215593, 0.0175886, 0.938263), 0.1, c(6, 0.5, 2, 9,
     0.5, 2), NULL,
                      c("popEVO", "0"))
#
# Parameter values  :
#
20 #   Delta     :  0.1            Rmax      :  2              Sb        :  0.05
#   Sj        :  1              Sm        :  2              Imax      :  1
#   q         :  0.938263       Sigma     :  0.5            T         :  0.1
#   p         :  1              Mu        :  0.01
#
25 # Index of bifurcation parameter #1                            : 6
# Index of structured population for evolutionary analysis     : 0
#
#           1:q    2:E[0]     3:b[0]      4:q' 5:I[0][0]  6:I[0][1] 7:I[0][2] 8:R0[0]   9:R0[1]  10:RHS norm
#
30 > output3$curvepoints
             V1        V2         V3        V4        V5         V6        V7      V8        V9          V10
 [1,] 0.9382634 0.2155930 0.01758857 0.9382634 0.1784407  0.4194359 0.3916582       1         1 1.461957e-07
 [2,] 0.9447576 0.2156234 0.01858414 0.9314420 0.1784377  0.4321497 0.3789306       1         1 7.201710e-10
 [3,] 0.9505209 0.2157066 0.01958022 0.9247685 0.1784293  0.4434929 0.3675496       1         1 4.059613e-09
35 <...output lines suppressed in this box...>
[74,] 1.0571668 0.2409761 0.08160450 0.5125038 0.1759024  0.5976217 0.2019346       1         1 9.543894e-12
[75,] 1.0579764 0.2414375 0.08237223 0.5061127 0.1758562  0.5984558 0.2008908       1         1 1.001310e-11
[76,] 1.0587853 0.2419027 0.08313670 0.4996828 0.1758097  0.5992899 0.1998452       1         1 1.059525e-11

40 > output4 <- PSPMequi("Indet_growth","PIP",c(output1$bifpoints[c(1,2,3,1)]),-0.1,c(6,0.5,2.0,6,0.5,2.0),
                     NULL,c("popEVO","0"))

Dynamic library file Indet_growthequi.so is up-to-date

45  9.38263361E-01  2.15592978E-01  1.75885671E-02  9.38263361E-01
  9.30859689E-01  2.15628723E-01  1.65937656E-02  9.45282279E-01
  9.22302855E-01  2.15749165E-01  1.56001074E-02  9.52520009E-01
<...output lines suppressed in this box...>
  5.13691984E-01  2.40890443E-01  7.02691650E-03  1.05701563E+00
50  5.04282772E-01  2.41569757E-01  6.99375827E-03  1.05820719E+00
  4.94835713E-01  2.42254183E-01  6.96170534E-03  1.05939157E+00

#
# Executing : PSPMequi("Indet_growth", "PIP", c(0.938263, 0.215593, 0.0175886, 0.938263), -0.1, c(6, 0.5, 2, 9,
     0.5, 2), NULL,
55                   c("popEVO", "0"))
#
# Parameter values  :
#
#   Delta     :  0.1            Rmax      :  2              Sb        :  0.05
60 #   Sj        :  1              Sm        :  2              Imax      :  1
#   q         :  0.938263       Sigma     :  0.5            T         :  0.1
#   p         :  1              Mu        :  0.01
#
# Index of bifurcation parameter #1                            : 6
65 # Index of structured population for evolutionary analysis     : 0
#
#           1:q    2:E[0]     3:b[0]      4:q' 5:I[0][0]  6:I[0][1] 7:I[0][2] 8:R0[0]   9:R0[1]  10:RHS norm
#
> output4$curvepoints
```

```
             V1        V2         V3        V4        V5         V6        V7      V8        V9          V10
 [1,] 0.9382634 0.2155930 0.01758857 0.9382634 0.1784407 0.41943589 0.3916582       1 1.0000001 1.461957e-07
 [2,] 0.9308597 0.2156287 0.01659377 0.9452823 0.1784371 0.40513171 0.4059461       1 1.0000000 2.609046e-10
 [3,] 0.9223029 0.2157492 0.01560011 0.9525200 0.1784251 0.38897893 0.4220442       1 1.0000000 9.018195e-09
<...output lines suppressed in this box...>
[38,] 0.5136920 0.2408904 0.00702692 1.0570156 0.1759110 0.10039077 0.6992045       1 1.0000001 1.027275e-07
[39,] 0.5042828 0.2415698 0.00699376 1.0582072 0.1758430 0.09835918 0.7009273       1 1.0000001 1.015845e-07
[40,] 0.4948357 0.2422542 0.00696171 1.0593916 0.1757746 0.09638593 0.7025894       1 1.0000001 1.184862e-07
```

The commands and output in the box above do not need further explanation, except that the first and the fourth column are the value of the resident and the mutant value of the parameter q, respectively. From the 8th and 9th column it can be verified that both the resident and the mutant type indeed attain $R_0 = 1$ in the equilibrium states computed. The demo script `"Indet_growth"` uses the first and fourth output columns, corresponding to the two bifurcation parameters, from the data matrices `output3$curvepoints` and `output4$curvepoints` that result from the first and second call to the function `PSPMequi` in the command box above to construct the pairwise invasibility plot (PIP).

Chapter 7

Simulating evolutionary dynamics

7.1 Theoretical background

In the context of adaptive dynamics the change over evolutionary time in a set of life history traits, characterizing the individuals of a population, can be described by the so-called *canonical equation* (Dieckmann and Law 1996). This canonical equation specifies a system of ordinary differential equation for the values of a trait vector $\mathbf{x} = (x_1, \ldots, x_n)$, assuming that the population size is large (infinite) and that evolution is limited by small mutation steps in the trait values. More specifically,

$$\frac{d\mathbf{x}}{dt} \;=\; n_e(\mathbf{x})\,\theta\,\boldsymbol{\Sigma}\,\frac{\partial s_{\mathbf{x}}(\mathbf{y})}{\partial \mathbf{y}}\bigg|_{\mathbf{y}=\mathbf{x}} \tag{7.1}$$

Here, $n_e(\mathbf{x})$ is the effective population size, θ the mutation probability per birth event, $\boldsymbol{\Sigma}$ the n–dimensional mutational variance–covariance matrix summarizing the distribution of mutations around the resident type $\mathbf{x}$ and $\partial s_{\mathbf{x}}(\mathbf{y})/\partial \mathbf{y}$ is the selection gradient (see also equation (6.1)). As discussed in section 6.1 the selection gradient is sign–equivalent with the following derivative of R_0:

$$\frac{\partial R_0(\mathbf{y}, \tilde{E}(\mathbf{x}))}{\partial \mathbf{y}}\bigg|_{\mathbf{y}=\mathbf{x}}$$

This partial derivative of R_0 with respect to life history parameters is the quantity that is used to analyse evolutionary fixed points of PSPMs, as explained in sections 6.1-6.3. Furthermore, we can assume that the effective population size $n_e(\mathbf{x})$ is proportional to the birth rate of a structured population, $\tilde{b}(\mathbf{x})$, for a given value of the trait vector. In other words, the evolutionary dynamics of the values of the life history parameters can be assumed to be proportional to the product of the population birth rate and the partial derivative of R_0:

$$\frac{d\mathbf{x}}{dt} \;\propto\; \tilde{b}(\mathbf{x})\,\boldsymbol{\Sigma}\,\frac{\partial R_0(\mathbf{y}, \tilde{E}(\mathbf{x}))}{\partial \mathbf{y}}\bigg|_{\mathbf{y}=\mathbf{x}} \tag{7.2}$$

Given that the software package routinely computes both $\tilde{b}(\mathbf{x})$ as well as $\partial R_0(\mathbf{y}, \tilde{E}(\mathbf{x}))/\partial \mathbf{y}$ while analyzing evolutionary fixed points in PSPMs, it is easy to understand that simulating the dynamics of the life history trait values over evolutionary time is a straightforward extension.

Hence, the `PSPManalysis` package contains in addition to the `PSPMdemo` and `PSPMequi` functions a function called `PSPMevodyn` to simulate the change in an arbitrary number of life history parameters over evolutionary time. As a starting point the function takes an ecological equilibrium state for a

particular set of parameters and computes both the partial derivative of R_0 with respect to the evolving parameters and the value of the population birth rate $\tilde{b}$ in equilibrium. Given these 2 quantities, it computes the value of the right-hand side of expression (7.2) that is proportional to the evolutionary rate of change in the life history parameters as determined by the canonical equation. Unless explicitly specified, the function assumes that the mutational variance—covariance matrix $\mathbf{\Sigma}$ equals the identity matrix. Finally, it uses the computed value of the evolutionary rate of change to derive new values for the evolving parameters using the Euler method for numerical integration of ordinary differential equations.

7.2 Arguments and output of the `PSPMevodyn` function

The use of the `PSPMevodyn` function will be illustrated with the same model as described in section 6.2 and analysed in section 6.3. In fact, the demo script `"Indet_growth"` that was already discussed in the previous chapter, performs at the end 2 computations of trait dynamics over evolutionary time. The demo script `"Indet_growth"` furthermore performs some plotting of the output data generated by these computations.

The general call to the `PSPMevodyn` function is shown in the command box below.

```
1 > output <- PSPMevodyn(modelname = NULL, startpoint = NULL, curvepars = NULL, evopars = NULL, covars = NULL,
                         parameters = NULL, options = NULL, clean = FALSE, force = FALSE, debug = FALSE)
```

The obligatory and optional arguments to the `PSPMevodyn` function are the following:

1. The first, obligatory argument to the function `PSPMevodyn` is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension `'.R'` or `'.h'` as part of the file name, the `PSPMevodyn` function will automatically try to locate the appropriate file, checking first for a file implemented in `C` (with an extension `'.h'`) and subsequently for a file implemented in `R` (with an extension `'.R'`). If both a file with an extension `'.h'` and a file with an extension `'.R'` are found, the program will use the first one. The program can be forced to use the file with an extension `'.R'` by including the extension explicitly as part of the filename. The `R`-commands to analyse the model specified in `Indet_growth.h` that will be used for the illustration below will therefore all take `"Indet_growth"` as their first argument. If the file specifying the PSPM can not be found in the current directory, the `PSPMevodyn` function will ask the user to search in the package directory for a model file with the specified name.

2. The second, obligatory argument is the initial point of the computation. This initial point should be close to an equilibrium point of the ecological dynamics. The initial point should be a (row) vector with the proper dimension, including as first elements the estimated equilibrium values for all the environment variables and the estimated values of the birth rate for all the structured populations in the model, followed by initial values for all parameters that are allowed to evolve over evolutionary time:

   `c(`<*environment variables*>,<*population birth rates*>,<*parameter 1*>,<*parameter 2*>`,...)`

   However, environment variables that have been explicitly specified with the program option `"envZE"` as having a zero equilibrium value and birth rates of populations that have been explicitly specified with the program option `"popZE"` to be in a zero equilibrium state (see the description of these options under point 7 below), should be omitted from this vector of initial values.

3. The third, obligatory argument to the `PSPMevodyn` function is a row vector consisting of 2 elements: (1) the maximum step size in evolutionary time during the integration of the canonical equation and (2) the maximum evolutionary time at which to stop the integration of the canonical equation.

4. The fourth, obligatory argument to the PSPMevodyn function determines which of the model parameters are allowed to evolve and at which limits further evolution of these parameter is prohibited. This information should be specified by a (row) vector, which for every evolving parameter should include a triplet of values specifying the index of the parameter, its minimum and its maximum value at which its evolution should stop. Therefore, in case of a single evolving parameter, the row vector is of the form:

   c(<*index 1*>,<*minimum 1*>,<*maximum 1*>)

   The first element of the vector indicates the index of the parameter in the array parameter to vary, while the final two elements of the array indicate the minimum and maximum value of the parameter. When two parameters are allowed to evolve, the row vector is of the form:

   c(<*index 1*>,<*minimum 1*>,<*maximum 1*>,<*index 2*>,<*minimum 2*>,<*maximum 2*>)

   With multiple evolving parameters the vector has to be extended with a triplet of values for each of these model parameters with the triplet specifying the index of the particular parameter as well as its minimum and the maximum value. The number of triplets should correspond with the number of initial values for the evolving parameters as specified in the second argument to the function. The integration of the canonical equation is halted before reaching the maximum integration time specified in the third argument to the function, whenever all evolving parameter have reached either their minimum or their maximum limit.

5. The fifth, optional argument of the PSPMevodyn function specifies the variance–covariance matrix $\mathbf{\Sigma}$ (see equation (7.1)). This argument can be specified as an $n \times n$ matrix or as a vector of length $n \cdot n$, where n equals the number of evolving parameters. The element (i, j) of the matrix (or equivalently the element $(i \cdot n + j)$ of the vector) should indicate how the selection gradient in trait j changes the value of trait i through genetic coupling. If the vector is not specified the matrix $\mathbf{\Sigma}$ is taken equal to the identity matrix.

6. The sixth, optional argument of the PSPMevodyn function is a (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (the length of the vector DefaultParameters in R, or the value of PARAMETER_NR in C). When of this length the values will replace the default values of the parameters that are listed in the model specification file. If the array used for this sixth argument is not of the correct length or when it is not specified at all, it will simply be ignored.

7. The seventh, optional argument of the PSPMevodyn function is a (row) vector of string elements, containing possible options that modify the behavior of the computational module. Most of the options require a value and hence occur as a pair of option name and option value. Only the "test" option (see below) occurs on its own. Options can be specified in any order, but the option value should always immediately follow after the option name. All option values refer to indices of either environment variables, structured populations or individual state variables. Notice, that this index value follows the C-convention of ordering arrays starting at 0 (as opposed to R where array indices start at 1). Multiple options can be included into the vector like:

   c("*name 1*", "*value 1*","*name 2*","*value 2*","*name 3*","*value 3*")

   Possible options are:

   i. Option pair c("popEVO", "i"): This option pair specifies the index of the structured population, whose life history parameters are evolving. If not specified, this index defaults to 0.

ii. Option pair `c("envZE", "i")`: This option pair can be specified several times as part of the option vector of strings. Including this option instructs the computational module to set the value of the environment variable with index `"i"` equal to 0 during the computations of the fixed point problem that determines the selection gradient in the evolving parameters. In addition, the equilibrium condition for this environment variable (as, for example, specified in code block 4.3.1.9 and 4.3.2.12) is ignored and hence not included as condition to hold in the particular equilibrium point. Notice that this can only occur for environment variables that are of the type `PERCAPITARATE` or `POPULATIONINTEGRAL` (see section 4.3.1.3 or section 4.3.2.12 above).

iii. Option pair `c("popZE", "i")`: This option pair can be specified several times as part of the option vector. Including this option forces the computational module to assume that the structured population with index `"i"` in the model is in a zero equilibrium state. This is the only way to compute an equilibrium with a zero equilibrium state for a particular population. Even if a value of 0 would be specified for the birth rate of a population as part of the initial point of the computation, the software would compute the equilibrium curve with a non-zero (non-trivial) equilibrium state for this population. Notice that if a structured population is forced to be in a zero equilibrium state by using the `"popZE"` option, a zero equilibrium state should also be enforced for all the environment variables that represent integrals over this population distribution (that are hence of the type `POPULATIONINTEGRAL`).

iv. Option pair `c("isort", "i")`: This option modifies the output of the equilibrium state of the populations that are stored in an output file with a name of the form `<Modelname>-EVODYN-<NNNN>.csb` (see below). By default the computational module reports the information about the stable population state distributions by subdividing the axis of the first state variable (the one with index `"0"`) in 100 subintervals of equal length and reporting the statistics for the cohort of individuals within each subinterval. By using the option `"isort"` the default choice to use the first individual state variable for this subdivision can be changed to the second, third, and so on. Therefore, passing `c("isort", "0")` as option vector to the `PSPMevodyn` function is the same as the default behaviour: the first individual state variable is used for the subdivision and ordering of the population state distribution, while passing `c("isort", "1")` would use the second individual state variable for this purpose. Also notice that the number of subdivisions of the individual state variable can be redefined by assigning the dimension `COHORT_NR` a value different from 100 (see section 9.3).

v. Option `c("test")`: The last possible option that can be passed to the `PSPMevodyn` function as part of the option vector is the `"test"` option. This invokes the computational module in testing mode, which implies that only a single integration of the individual life history is carried out and no iteration to locate a fixed point of a set of equations is performed. In testing mode the computational module reports on the dynamics of the individual state variables, the survival, the cumulative impact on the environment and the expected number of offspring produced by an individual during its different life stage as well as over its entire life. Testing mode is very useful to discover whether or not the model implementation gives sensible results or not.

Three other optional arguments can be passed to the `PSPMevodyn` function: `clean`, `force` and `debug`. These are all boolean arguments that hence have to be passed to the `PSPMevodyn` function as *<option name>*`=TRUE` or *<option name>*`=FALSE`, the latter being the default value of all options (Specifying these options as argument is hence only useful when setting them equal to `TRUE`). Unlike the previous arguments, which all modify the computations to be performed, these options modify the behavior of the `PSPMevodyn` function itself, in particular the compilation of the model specific file into a dynamic library module that can be executed from `R`. Also unlike all the previous arguments that can

be passed, these arguments can be passed in any order and at any position, the PSPMevodyn function will filter these 3 optional arguments from the argument list before passing the filtered argument list to the computational routine.

- Option clean: When clean=TRUE is passed as argument, this argument instructs the PSPMevodyn function to delete all result files that have been generated during previous calculations with the model. These result files have names of the form <Modelname>-<Type>-<NNNN>.err, <Modelname>-<Type>-<NNNN>.csb and <Modelname>-<Type>-<NNNN>.out, in which <Modelname> refers to the name of the model, <Type> refers to the type of computation that has been performed, which in the case of PSPMevodyn equals EVODYN, and <NNNN> is a unique number that distinguishes consecutive computations of the same type of curve with the same model. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function PSPMclean() to delete all .bif, .err, .csb and .out files and/or all executable files that are present in the current working directory.

- Option force: When force=TRUE is passed as argument, it instructs the PSPMevodyn function to force re-compilation of the model specific file into a dynamic library module that can be executed by R. This option will usually not be needed by normal users, as the PSPMevodyn function automatically recompiles the computational module when the model specific file with an '.h' extension is more recently changed than the compiled dynamic library file. However, if for some unclear reason this automatic recompilation fails, the force option can be used to initiate re-compilation.

- Option debug: When debug=TRUE is passed as argument, it instructs the PSPMevodyn function to turn on debugging flags while compiling the model specific file into a dynamic library module. This option can be useful to detect programming mistakes in the model-specific file that are otherwise hard to track down. The downside is that depending on the version of R that is used, turning on debugging flags during compilation may generate a lot of output, including warnings about standard files of the operating system that are perfectly correct. It is hence not so easy to spot among all these messages the warnings that relate to the model-specific code that has been implemented.

The computational module generates on execution a single list object as output with 2 member elements (see the help page on PSPMevodyn using ?PSPMevodyn). The first element of the output list, output$curvepoints contains the numerical information of the points along the computed curve. This variable output$curvepoints is a matrix, in which each row represents one solution point along the curve. The columns contain the evolutionary time value, the equilibrium value of all environment variables, the equilibrium value for the birth rate of all structured populations in the problem, the current value of the evolving parameter(s), the equilibrium value of all interaction variables defined in the routine Impact(), the per capita growth rate of all environment variables for which this is relevant (those of the type PERCAPITARATE), for each of the structured populations the expected number of offspring produced by an individual during its lifetime (R_0) and finally the norm of the right-hand side of the system of equations that is solved to obtain the ecological equilibrium. The latter quantity (referred to as RHS norm) measures how close the computed equilibrium point is to the true solution.

The second member element of the output list, output$curvedesc, contains the description of the executed calculation, which includes the command-line that is used for the invocation of the computational routine, the values of all parameters used for the current computation and a header line indicating the meaning of all the output variables produced by the computational module. This textual information is also printed to the R console at the end of calculations. In fact, the PSPMevodyn function prints its report on the calculations by execution of the statement cat(output$curvedesc, sep='\n').

7.3 An example session using the PSPMevodyn function

The demo script "Indet_growth" illustrates the use of the PSPMevodyn function by simulating the evolutionary dynamics in the parameter q, the scaling power in the model implemented in Indet_growth.h that relates the resource ingestion rate to individual body size (see section 6.2). The particular call to the PSPMevodyn function is shown in the following R command box:

Command box 7.3.A

```
> output1 <- PSPMevodyn("Indet_growth", c(0.22, 0.03554, 1.0), c(0.05, 10), c(6, 0.5, 1.5), options=c("popEVO",
    "0"))

Building executable Indet_growthevodyn.so ...

<...compilation output lines suppressed in this box...>

 0.005000    2.20000000E-01  3.55373787E-02  1.00000000E+00
 0.010000    2.18793851E-01  3.19778211E-02  9.92635669E-01
 0.015000    2.18089189E-01  2.97666950E-02  9.87408855E-01
<...output lines suppressed in this box...>
 2.581562    2.15592977E-01  1.75885665E-02  9.38263363E-01
 2.631562    2.15592977E-01  1.75885665E-02  9.38263362E-01
 2.681562    2.15592977E-01  1.75885664E-02  9.38263362E-01

#
# Executing : PSPMevodyn("Indet_growth", c(0.22, 0.03554, 1), c(0.05, 10), c(6, 0.5, 1.5), NULL, NULL,
    c("popEVO", "0"))
#
# Parameter values  :
#
#   Delta     :  0.1            Rmax      :  2              Sb        :  0.05
#   Sj        :  1              Sm        :  2              Imax      :  1
#   q         :  1              Sigma     :  0.5            T         :  0.1
#   p         :  1              Mu        :  0.01
#
# Index of structured population for evolutionary dynamics   : 0
# Index of evolution parameter #0                            : 6
#
#  1:Evol.time    2:E[0]     3:b[0]       4:q 5:I[0][0] 6:I[0][1] 7:I[0][2]   8:R0[0]   9:RHS norm
#
> output1$curvepoints
            V1        V2         V3        V4        V5        V6        V7        V8           V9
 [1,] 0.005000 0.2200000 0.03553738 1.0000000 0.1780000 0.5323024 0.2767885 1.0000000 9.586909e-09
 [2,] 0.010000 0.2187939 0.03197782 0.9926357 0.1781206 0.5210074 0.2886318 1.0000000 8.771789e-09
 [3,] 0.015000 0.2180892 0.02976670 0.9874089 0.1781911 0.5124553 0.2975042 1.0000000 3.522374e-09
<...output lines suppressed in this box...>
[71,] 2.581562 0.2155930 0.01758857 0.9382634 0.1784407 0.4194359 0.3916581 0.9999998 2.308560e-07
[72,] 2.631562 0.2155930 0.01758857 0.9382634 0.1784407 0.4194359 0.3916581 0.9999998 2.308562e-07
[73,] 2.681562 0.2155930 0.01758857 0.9382634 0.1784407 0.4194359 0.3916581 0.9999998 2.308564e-07
```

Starting from an (approximate) equilibrium resource density of 0.22 and an (approximate) equilibrium birth rate value of 0.03554 for an initial parameter value $q = 1.0$ the evolutionary dynamics is simulated from $t = 0$ (this starting time is always taken equal to 0) till $t = 2.681562$. The simulation stops at this time point, because the evolution in q has converged to a fixed value and q is not going to change any further. This final value of q hence represents a stable and attracting evolutionary state (CSS). In general, the computation will be stopped whenever all evolving parameters have stabilized at a constant value.

The starting point of the computation is contained in the second argument to the function PSPMevodyn, as shown above, whereas the first argument defines the basename of the file with the model implementation "Indet_growth". The third argument to the function PSPMevodyn sets the maximum evolutionary time step to 0.05 and the maximum time at which to stop the evolutionary

simulation to 10.0, but the latter is never reached because of the convergence to an evolutionarily constant q value. The fourth argument to the function `PSPMevodyn` contains a single triplet of values, given that only a single parameter is allowed to evolve, defining the index of the parameter q in the `parameter` array as defined in `Indet_growth.h` and its minimum and maximum value at which to stop the computations. The fifth and sixth argument are left undefined, which implies that the variance–covariance matrix $\mathbf{\Sigma}$ (refer to equation (7.1)) defaults to the identity matrix and default values are used for all non-evolving model parameters. These default values are defined in the file `Indet_growth.h`. The final argument to the function `PSPMevodyn`, the option vector, defines the index of the structured population, in which the evolutionary dynamics takes place, equal to 0, but as discussed before, specifying this option is superfluous as the option `"popEVO"` is equal to 0 by default.

During the computations the program reports the current value of the evolutionary time, the ecological equilibrium values of the resource density and the population birth rate and the current value of the evolving parameter q. In the `R` command box above the values of the computed points along the evolutionary trajectory are saved in the output list element `output1$curvepoints`, the contents of which are inspected after the `PSPMevodyn` function finishes and it has printed out the textual information about the computation. The demo script `"Indet_growth"` uses the data contained in the first and the fourth column of `output1$curvepoints` to plot the time course of evolutionary change in the parameter q.

In the following `R` command box, a similar trajectory of the evolutionary dynamics is computed starting from the same ecological equilibrium, but now both the parameter q and the parameter p, which relates the maintenance costs to individual body size (see section 6.2), are allowed to evolve. To that end, the starting point of the computation is extended with an initial value for the parameter p (1.0) and the fourth argument of the `PSPMevodyn` function is extended with a triplet of values that indicate the index of the parameter p in the `parameter` array as defined in `Indet_growth.h` and its minimum and maximum value at which to stop the computations. In addition, the maximum integration time at which to stop the evolutionary computations is increased to 100. Otherwise, the command line of this computation is identical to the one shown in `R` command box 7.3.A.

Command box 7.3.B

```
> output2 <-
    PSPMevodyn("Indet_growth",c(0.22,0.03554,1.0,1.0),c(0.05,100),c(6,0.5,1.5,9,0.5,1.5),options=c("popEVO","0"))
Dynamic library file Indet_growthevodyn.so is up-to-date

 0.005000    2.20000000E-01  3.55373787E-02  1.00000000E+00    1.00000000E+00
 0.010000    2.18004640E-01  2.95832099E-02  9.92635669E-01    1.00669514E+00
 0.015000    2.17277498E-01  2.70926149E-02  9.88746582E-01    1.01012886E+00
<...output lines suppressed in this box...>
59.960312    2.13281602E-01  1.20360784E-02  4.99880243E-01    5.34824255E-01
60.010312    2.13281602E-01  1.20360784E-02  4.99880243E-01    5.34824254E-01
60.060312    2.13281602E-01  1.20360784E-02  4.99880243E-01    5.34824254E-01

#
# Executing :
    PSPMevodyn("Indet_growth",c(0.22,0.03554,1,1),c(0.05,100),c(6,0.5,1.5,9,0.5,1.5),NULL,NULL,c("popEVO","0"))
#
# Parameter values  :
#
#   Delta     :  0.1           Rmax      :  2             Sb        :  0.05
#   Sj        :  1             Sm        :  2             Imax      :  1
#   q         :  1             Sigma     :  0.5           T         :  0.1
#   p         :  1             Mu        :  0.01
#
# Index of structured population for evolutionary dynamics   : 0
# Index of evolution parameter #0                            : 6
# Index of evolution parameter #1                            : 9
#
```

```
#     1:Evol.time    2:E[0]     3:b[0]       4:q       5:p 6:I[0][0] 7:I[0][1] 8:I[0][2] 9:R0[0]  10:RHS norm
#
> output2$curvepoints
               V1        V2         V3        V4        V5        V6        V7        V8      V9          V10
   [1,]  0.005000 0.2200000 0.03553738 1.0000000  1.000000 0.1780000 0.5323024 0.2767885       1 9.586909e-09
   [2,]  0.010000 0.2180046 0.02958321 0.9926357  1.006695 0.1781995 0.5129883 0.3003329       1 3.986293e-09
   [3,]  0.015000 0.2172775 0.02709261 0.9887466  1.010129 0.1782722 0.5013552 0.3134773       1 3.482050e-09
<...output lines suppressed in this box...>
[1216,] 59.960312 0.2132816 0.01203608 0.4998802 0.5348243 0.1786718 0.3554964 0.3474220 0.99999 7.866359e-07
[1217,] 60.010312 0.2132816 0.01203608 0.4998802 0.5348243 0.1786718 0.3554964 0.3474220 0.99999 7.866359e-07
[1218,] 60.060312 0.2132816 0.01203608 0.4998802 0.5348243 0.1786718 0.3554964 0.3474220 0.99999 7.866356e-07
```

The output in the command box above shows that again the evolutionary dynamics are halted before the maximum time (100) is reached. As soon as an evolving parameter (here the parameter q) drops below its minimum value or exceeds its maximum value, as specified in the fourth argument to the function `PSPMevodyn`, it is stopped from evolving further, which in the case of the computation shown above leads to convergence to a constant value of the second evolving life history parameter p. This convergence ultimately halts the computation. The computation is therefore in this case stopped because no further evolution occurs, but computations will also stop whenever all evolving parameters have reached either their minimum or their maximum limit.

7.4 Output files generated by the `PSPMevodyn` function

The computational module that is produced by the `PSPMevodyn` function generates 3 output files. The name of these files is always of the form `<Modelname>-EVODYN-<NNNN>.<ext>`, in which `<Modelname>` is the same as the name of the file specifying the model excluding its `'.R'` or `'.h'` extension, `<NNNN>` is a 4-digit number that is unique for the current computation and `.<ext>` is the extension, which can be either `.err`, `.csb` or `.out`. The unique number distinguishes the same types of curve computations for the same model from each other. The number is obtained by considering increasing values of `<NNNN>` (i.e., `0000`, `0001`, `0002` and so forth) and testing whether result files with the particular index are already present. The program uses the first value of `<NNNN>` that is not in use.

The file called `<Modelname>-EVODYN-<NNNN>.err` that is generated during the computations contains information about the numerical progress of the computations. It reports details on the steps taken during the Newton iteration, the convergence to the solution, as well as information about the steps taken along the curve that is being computed. This file can be informative in case the computation of a particular curve stops for unknown reasons, but is otherwise of little use.

The output file called `<Modelname>-EVODYN-<NNNN>.out` holds the same information as is contained in the two elements of the output list returned by the `PSPMevodyn` function, `output$curvepoints` and `output$curvedesc` (see the help page on `PSPMevodyn` using `?PSPMevodyn`). The first lines of this file all start with a `'#'` sign and contain the information about the run performed, which is also contained in `output$curvedesc` and can be listed by the statement `cat(output$curvedesc, sep='\n')`. Following this descriptive header the file contains columns with computational results that are also contained in the variable `output$curvepoints` (see, for example, `R` command box 7.3.A). In fact, the two elements of the output list, `output$curvepoints` and `output$curvedesc`, are generated by reading the contents of the file `<Modelname>-EVODYN-<NNNN>.out` from disk after the computations have ended, storing all lines that start with a `'#'` sign into a single string variable `output$curvedesc`, while storing the information on all other lines into the data matrix `output$curvepoints`.

The file called `<Modelname>-EVODYN-<NNNN>.csb` contains for every curve point that has been computed information on the parameters, for which the point has been computed, the equilibrium values of all environment variables and the stable distribution of all structured populations in the model. This is a binary file, the content of which can be accessed from `R` using the function `csbread`. For

example, the file `Indet_growth-EVODYN-0000.csb` is generated by the invocation of the `PSPMevodyn` function in `R` command box 7.3.A. Its contents can be listed by:

Command box 7.4.A

```
> csbread("Indet_growth-EVODYN-0000.csb")

States in file Indet_growth-EVODYN-0000.csb:

    1: State-5.000000E-03
    2: State-1.000000E-02
    3: State-1.500000E-02
<...output lines suppressed in this box...>
   71: State-2.581562E+00
   72: State-2.631562E+00
   73: State-2.681562E+00
```

The structure called `State-1.500000E-02` contains the population state in the ecological equilibrium that occurred at time point $t = 0.015$ during the simulation of evolutionary dynamics, as its name suggests. Its contents can be read into the workspace by issuing the command `csbread("Indet_growth-EVODYN-0000.csb",3)` or `csbread("Indet_growth-EVODYN-0000.csb","State-1.500000E-02")`.

Loading this state into the `R` workspace reveals it to be a list containing various arrays of numbers, as shown in the following box:

Command box 7.4.B

```
> popstate <- csbread("Indet_growth-EVODYN-0000.csb", "State-1.500000E-02")
> popstate
$EvoTime
[1] 0.015

$EvoPars
[1] 0.9874089

$Parameters
 [1] 0.1000000 2.0000000 0.0500000 1.0000000 2.0000000 1.0000000 0.9874089 0.5000000 0.1000000 1.0000000
[11] 0.0100000

$Environment
[1] 0.2180892

$Pop00_BirthStates
     Istate00 Istate01
[1,]        0     0.05

$Pop00
            Density   Istate00   Istate01
  [1,] 5.571417e-01   10.00401 0.05718031
  [2,] 4.528618e-01   30.72728 0.07462823
  [3,] 3.680999e-01   51.45054 0.09665210
<...output lines suppressed in this box...>
 [98,] 1.037446e-09 2020.16080 1.98742578
 [99,] 8.432681e-10 2040.88406 1.98758088
[100,] 6.854340e-10 2061.60733 1.98773223
```

The first element of the list (called `$EvoTime`) representing the population state `State-1.500000E-02` is the value of the evolutionary time t at which the current population state occurs. The second element, an array called `$EvoPars`, contains the values at evolutionary time t of all the evolving model parameters. The third element, an array called `$Parameters`, contains the values of all the model parameters for which the population state has been computed, while the fourth member of the list contains the equilibrium values of all environment variables. The two subsequent arrays in the list characterise the stable population distribution, of which the first (called `$Pop00_BirthStates`)

specifies the state at birth of the individuals. The other (called `$Pop00`) is a two-dimensional array characterising the population distribution in equilibrium with the first column `$Pop00[,1]` representing the density profile of the equilibrium population and the subsequent columns `$Pop00[,2]` and `$Pop00[,3]` representing the average values of the individual state variables with index 0 and 1 (corresponding to individual age and body size in the model implemented in `Indet_growth.h`), as shown in the `R` command box above. If individuals are characterized by more than two individual state variables, the values of these follow in additional columns of the two-dimensional array `$Pop00`. The `R` command box above also illustrates that the dimension of the array `$Pop00` indicates that the population is represented by 100 cohorts of individuals (see section 9.3 for the option to change this number). The number of individuals in cohort i is given by the array element `$Pop00[`i`,1]`, while the average value of the individual state variable with index 0 and 1 (average age and average size in the current model) are given by `$Pop00[`i`,2]` and `$Pop00[`i`,3]`, respectively.

Chapter 8

Simulating the individual life history

8.1 Simulating individual life histories in specific environments

To analyse the factors and mechanisms that lead to specific changes in model equilibria with a change in parameters it is often necessary to compute the life history trajectory of an individual organism at a specific set of environmental conditions. Although to some extent this information can be extracted from the population state file (the `.csb` file) that is generated as output, the package also contains a separate function `PSPMind` that computes the individual life history, given a particular set of values for the environmental variables. In principle, the `PSPMind` function only requires the values of environmental variables as input. However, this input vector of environmental variables can be extended with the values of the birth rates of all structured populations in the model. Providing values of the population birth rates will scale the output of the function `PSPMind` with these birth rates, which is useful if the `PSPMind` function is used for generating an initial state for simulating the ecological dynamics of the model, as discussed in chapter 5.

The output of `PSPMind` function is a structure with similar contents as the structures that are normally stored in the `.csb` file (see section 3.4.2, 4.4.5, 5.3 or 7.4 for a discussion of the `.csb` file). In fact, the program generates such a file with a name `<Modelname>-IND-<NNNN>.csb` which contains exactly one population state. As before, `<Modelname>` is the same as the name of the file specifying the model excluding its `'.R'` or `'.h'` extension and `<NNNN>` is a 4-digit number that is unique for the current computation.

8.1.1 Arguments and output of the `PSPMind` function

The general call to the `PSPMind` function is shown in the command box below.

```
1 > output <- PSPMind(modelname = NULL, environment = NULL, parameters = NULL, options = NULL,
                      clean = FALSE, force = FALSE, debug = FALSE)
```

The obligatory and optional arguments to the `PSPMind` function are the following:

1. The first, obligatory argument to the function `PSPMind` is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension `'.R'` or `'.h'` as part of the file name, the `PSPMevodyn` function will automatically try to locate the appropriate file, checking first for a file implemented in `C` (with an extension `'.h'`) and subsequently for a file implemented in `R` (with an extension `'.R'`). If both a file with an extension `'.h'` and a file with an extension `'.R'` are found, the program will use the first one. The program can be forced to use the file with an extension `'.R'` by including the extension explicitly as part of the filename. The `R`-command to simulate the life history with the model specified in `PNAS2002.h`, which will be used for the illustration below, therefore takes `"PNAS2002"` as its first argument. If the file

specifying the PSPM can not be found in the current directory, the `PSPMind` function will ask the user to search in the package directory for a model file with the specified name.

2. The second, obligatory argument is a (row) vector containing the values of the environmental variables, for which to compute the individual life history:

   `c(`*<environment variable 1>*`,`*<environment variable 2>*`,....)`

   The number of values specified in this vector should equal the number of environmental variables in the model, that is, the length of the variable `EnvironmentState` for models implemented in `R` (see code block 4.3.1.3) and the value of the constant `ENVIRON_DIM` for models implemented in `C` (see code block 4.3.2.1). Notice, that this vector therefore does not contain any parameter values.

   Alternatively, the vector in this second argument can be extended with the birth rates of the structured population model. In this case, the argument has the form:

   `c(`*<environment variable 1>*`,`*<environment variable 2>*`,....,`*<birth rate 1>*`,`*<birth rate 2>*`,....)`

   Notice that as many birth rates have to be specified as there are structured populations in the model (the value of the element `PopulationNr` in the vector `PSPMdimensions` for models implemented in `R`, see code block 4.3.1.1, or the value of `POPULATION_NR` models implemented in `C`, see code block 4.3.2.1). The effect of specifying these birth rates is that the value of the individual survival produced as output by the model (see command box 8.1.2 below) will be multiplied by the birth rates of the particular structured population model. This allows for arbitrary scaling of the number of individuals in the output, which is useful if the function `PSPMind` is used for producing an initial state for the function `PSPMecodyn` (see chapter 5). If the birth rates are not specified the program assumes a default value of these rates equal to 1.

3. The third, optional argument of the `PSPMind` function is a (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (the length of the vector `DefaultParameters` in `R`, or the value of `PARAMETER_NR` in `C`). When of this length the values will replace the default values of the parameters that are listed in the model specification file. If the array used for this third argument is not of the correct length or when it is not specified at all, it will simply be ignored.

4. The fourth, optional argument of the `PSPMind` function is a (row) vector of string elements, containing possible options that modify the behavior of the computational module. The `PSPMind` function only recognizes a single option `isort`. Hence, this fourth argument is either left unspecified (or, equivalently, specified as `options = NULL`, which is the default) or takes the form:

   ```
   c("isort", "i")
   ```

   This option modifies the output of the equilibrium state of the populations that are stored in the output file with a name of the form `<Modelname>-IND-<NNNN>.csb` (see below). By default the computational module reports the information about the stable population state distributions by subdividing the axis of the first state variable (the one with index `"0"`) in 100 subintervals of equal length and reporting the statistics for the cohort of individuals within each subinterval. By using the option `"isort"` the default choice to use the first individual state variable for this subdivision can be changed to the second, third, and so on. Therefore, passing `c("isort", "0")` as option vector to the `PSPMind` function is the same as the default behaviour: the first individual state variable is used for the subdivision and ordering of the population state distribution, while passing `c("isort", "1")` would use the second individual state variable for this purpose. Also notice that the number of subdivisions of the individual state variable can be redefined by assigning the dimension `COHORT_NR` a value different from 100 (see section 9.3).

Three other optional arguments can be passed to the `PSPMind` function: `clean`, `force` and `debug`. These are all boolean arguments that hence have to be passed to the `PSPMind` function as <*option name*>`=TRUE` or <*option name*>`=FALSE`, the latter being the default value of all options (Specifying these options as argument is hence only useful when setting them equal to `TRUE`). Unlike the previous arguments, which all modify the computations to be performed, these options modify the behavior of the `PSPMind` function itself, in particular the compilation of the model specific file into a dynamic library module that can be executed from `R`. Also unlike all the previous arguments that can be passed, these arguments can be passed in any order and at any position, the `PSPMind` function will filter these 3 optional arguments from the argument list before passing the filtered argument list to the computational routine.

- Option `clean`: When `clean=TRUE` is passed as argument, this argument instructs the `PSPMind` function to delete all result files that have been generated during previous calculations with the model. These result files have names of the form `<Modelname>-<Type>-<NNNN>.csb`, in which `<Modelname>` refers to the name of the model, `<Type>` refers to the type of computation that has been performed, which in the case of `PSPMind` equals `IND`, and `<NNNN>` is a unique number that distinguishes consecutive computations of the same type of curve with the same model. Deleting all the output files from previous computations and/or the compiled program executables that the package has generated can also be done separately. The package implements a function `PSPMclean()`, taking no arguments, to delete all `.bif`, `.err`, `.csb` and `.out` files and/or all executable files that are present in the current working directory.

- Option `force`: When `force=TRUE` is passed as argument, it instructs the `PSPMind` function to force re-compilation of the model specific file into a dynamic library module that can be executed by `R`. This option will usually not be needed by normal users, as the `PSPMind` function automatically recompiles the computational module when the model specific file with an `'.R'` or `'.h'` extension is more recently changed than the compiled dynamic library file. However, if for some unclear reason this automatic recompilation fails, the `force` option can be used to initiate re-compilation.

- Option `debug`: When `debug=TRUE` is passed as argument, it instructs the `PSPMind` function to turn on debugging flags while compiling the model specific file into a dynamic library module. This option can be useful to detect programming mistakes in the model-specific file that are otherwise hard to track down. The downside is that depending on the version of `R` that is used, turning on debugging flags during compilation may generate a lot of output, including warnings about standard files of the operating system that are perfectly correct. It is hence not so easy to spot among all these messages the warnings that relate to the model-specific code that has been implemented.

The output of function is discussed in the example below.

8.1.2 An example using the `PSPMind` function

To illustrate the use of the `PSPMind` function, I will consider the example as discussed in section 4.4.5 and shown in command box 4.4.5.B. There, an equilibrium population state called `State_4_042258E_04` pertaining to the parameter value $R_{max} = 4.042258 \cdot 10^{-4}$, was read from the population state (`.csb`) file using the function `csbread`. The vector of environmental variables for the computed model equilibrium equals `c(2.533511e-04, 1.335974e-04, 4.008016e-06)`. Computing the individual life history for this particular environmental state with `PSPMind` gives the following result:

Command box 8.1.2

```
> output <- PSPMind("PNAS2002",c(2.533511e-04, 1.335974e-04, 4.008016e-06), options = c("isort", "1"),
    clean=TRUE, force=TRUE)

Building executable PNAS2002ind.so using sources from /Users/andre/programs/PSPM analysis ...

<...compilation output lines suppressed in this box...>

                              Istate[0]  Istate[1]  Survival  R0  Impact[0]  Impact[1]  Impact[ 2]  Impact[3]
Pop. #0 - Bstate 0 - (Final):   1237.26    283.066     1E-09   1  0.0512525  0.0136153   0.0370566   0.625002

> output
$Parameters
 [1] 1.0e-01 3.0e-04 7.0e+00 2.7e+01 1.1e+02 3.0e+02 9.0e-06 1.0e-04 ... 3.0e-03 1.0e-02 5.0e+03 1.0e-01 5.0e-01
     1.0e-02

$Environment
[1] 2.533511e-04 1.335974e-04 4.008016e-06

$Pop00
           Survival    Istate00  Istate01    Impact00    Impact01     Impact02    Impact03         R0
  [1,] 1.000000e+00    0.000000   7.00000 0.000000000 0.000000000 0.0000000000 0.000000000 0.00000000
  [2,] 3.221467e-01    1.674051   9.76066 0.006303773 0.004973069 0.0000000000 0.000000000 0.00000000
  [3,] 1.025926e-01    3.365087  12.52132 0.009951990 0.008797279 0.0000000000 0.000000000 0.00000000
<...output lines suppressed in this box...>
 [99,] 3.652493e-07  647.198013 277.54468 0.050983548 0.013615309 0.0370566144 0.617830389 0.99193394
[100,] 1.206611e-07  757.957444 280.30534 0.051163073 0.013615309 0.0370566144 0.622602248 0.99731968
[101,] 1.000000e-09 1237.256038 283.06600 0.051252524 0.013615309 0.0370566144 0.625001749 1.00000322
```

Passing the option vector `c("isort", "1")` as argument implies that the interval between the minimum and maximum value of the individual state variable with index 1 (here ranging between 7 and 283.1) is subdivided into 100 subintervals and that life history values are provided at these 100 intermediate values of that state variable. At these points in the life history the output contained in `$Pop00` shows the individual survival, the values of the individual state variables, the values of the different *cumulative* impacts of the individual as well as the value of its lifetime reproductive success R_0. If the model contains multiple structured populations, the individual life histories of individuals in the other populations will follow as `$Pop01`, `$Pop02`, etc.

Do note, however, that the function `PSPMind` will simulate the individual life history for the default values of the parameters, unless the `parameters` argument is passed to the function. Therefore, technically the life history shown above pertains to a parameter value of $R_{max} = 3.0 \cdot 10^{-4}$ (default value) as opposed to $R_{max} = 4.042258 \cdot 10^{-4}$, for which the equilibrium environmental variables were calculated. In this particular case this does not matter because the parameter R_{max} represents the resource productivity which does not affect the individual life history at all. But in case the bifurcation parameter does influence the individual life history make sure to pass the proper parameter vector as an argument to the function `PSPMind`.

Chapter 9

Additional information

9.1 Multiple states at birth

The previous chapters focused mainly on models that assume that all newborn individuals have the same, unique state-at-birth. In the Medfly model, presented in section 3.2 to discuss the implementation of a model for demographic analysis, individual age was the only *i-state* variable, which obviously equals 0 for all individuals at birth. In the PNAS model, presented in section 4.2 to discuss the implementation of a model for equilibrium analysis, all individuals were assumed to be born with the same length at birth $\ell = \ell_b$. The package contains, however, also 4 files (`Indet_growth_5bs.h`, `KlanjscekDEB2.h`, `Medfly_periodic.h` and `PNAS2002_5bs.h`) that implement models, in which individuals have different states-at-birth. These models will not be discussed extensively. Instead, the following sections will only briefly present some details about their implementation and usage, which are specific to the multiple states-at-birth. These models are all implemented in `C`, as implementing a PSPM with multiple states at birth will be too slow for all practical purposes.

The implementation of a model with multiple states-at-birth differs in at least 3 aspects from a model with a unique state-at-birth:

- The number of possible states at birth has to be defined larger than 1.
- The values of the *i-state* variables at birth have to be defined separately for the different states-at-birth.
- Not only the number of offspring produced has to be specified, but also the state-at-birth of the offspring has to be specified.
- In addition, the state-at-birth of an individual may influence the threshold value separating consecutive stages, the development and discrete changes in the individual state variables, the fecundity, the mortality and the impact of the individual on its environment (the latter only in case of equilibrium analysis of non-linear models). If this is the case, the values assigned in the routines `IntervalLimit()`, `Growth()`, `DiscreteChanges()`, `Fecundity()`, `Mortality()`, and possibly `Impact()` (see sections 4.3.2.6-4.3.2.11 for an explanation about these functions) will be dependent on the state-at-birth as well.

This last aspect is, however, not absolutely necessary, whereas the aspects 1-3 mentioned above are.

9.1.1 Demographic analysis with multiple states-at-birth

9.1.1.1 Two different offspring body sizes

The file `KlanjscekDEB2.h` implements a model, in which the life history of the individuals is described by a dynamic energy budget (DEB) model. The model is a variant of the model implemented in the file `KlanjscekDEB.h`, which assumes that all individuals at birth have the same size at birth V_b. In contrast, the model implemented in the file `KlanjscekDEB2.h` is based on an assumption that two types of offspring are produced: small offspring with a body size $0.7 \cdot V_b$ and large offspring with a body size $1.3 \cdot V_b$. Both models are discussed in detail in De Roos (2008), the model implemented in `KlanjscekDEB.h` on page 5-7 and the model with two types of offspring (implemented in `KlanjscekDEB2.h`) on page 13-14 of De Roos (2008). The implementations of these two models can be listed by executing the commands `showpspm("KlanjscekDEB")` and `showpspm("KlanjscekDEB2")`. The script `KlanjscekDEB` that is included with the package as a demo carries out the demographic analysis for both models and graphs the results as a function of food density in the environment, which can be compared with Figure 2 in De Roos (2008).

As shown in the code box below, which contains a snippet from the model implemented in `KlanjscekDEB2.h`, the number of possible states-at-birth should be defined larger than 1 in the routine `SetBirthStates()`, in this particular model `BirthStates[0]` is set equal to 2.

Code block 9.1.1.1.A

```
/*
 *=========================================================================
 *      DEFINITION OF THE LIFE HISTORY MODELS FOLLOWS BELOW
 *=========================================================================
 * Specify the number of states at birth for the individuals in all structured
 * populations in the problem in the vector BirthStates[].
 *=========================================================================
 */

void SetBirthStates(int BirthStates[POPULATION_NR], double E[])
{
  BirthStates[0] = 2;

  return;
}
```

The values of the *i-state* variables in the different states-at-birth have to be defined separately. This means that in the routine `StateAtBirth()` the assignment of the values to the variables `istate[][]` has to be made conditional on the value of the index of the state-at-birth `BirthStateNr`. The model implemented in the file `KlanjscekDEB2.h` has 2 states-at-birth $\{\phi_1, \phi_2\}$. In the code box below it is shown that the state-at-birth ϕ_1 (with index `BirthStateNr` $= 0$) corresponds to the small-sized offspring with body size $0.7 \cdot V_b$, while the state-at-birth ϕ_2 (`BirthStateNr` $= 1$) corresponds to the large-sized offspring with body size $1.3 \cdot V_b$.

Code block 9.1.1.1.B

```
/*
 *=========================================================================
 * Specify all the possible states at birth for all individuals in all
 * structured populations in the problem. BirthStateNr represents the index of
 * the state of birth to be specified. Each state at birth should be a single,
 * constant value for each i-state variable.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 *=========================================================================
```

```
 */

void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
{
  if (BirthStateNr == 0)
    {
      AGE    = 0.0;
      VOLUME = 0.7*VB;
      Q      = 0.0;
      H      = 0.0;
    }
  else
    {
      AGE    = 0.0;
      VOLUME = 1.3*VB;
      Q      = 0.0;
      H      = 0.0;
    }

  return;
}
```

Finally, the last part of the code that has to be changed in case of multiple states-at-birth is the assignment of fecundity to different states-at-birth. The model implemented in the file `KlanjscekDEB2.h` assumes that individuals with a different state-at-birth differ in their offspring production. More specifically, individuals that are born with a small size ($V = 0.7 \cdot V_b$) are assumed to invest 2/3 of the energy that they have available for reproduction on producing small offspring (i.e. with $V = 0.7 \cdot V_b$) and 1/3 of the reproductive energy producing large-sized offspring with $V = 1.3 \cdot V_b$. Vice versa, individuals that are born with a large body size ($V = 1.3 \cdot V_b$) are assumed to invest 2/3 of the energy that they have available for reproduction on producing large offspring (i.e. with $V = 1.3 \cdot V_b$) and 1/3 of the reproductive energy producing small-sized offspring with $V = 0.7 \cdot V_b$. Parents bias their energetic investment into reproduction therefore toward producing offspring with the same size at birth as they were born with themselves. These different energetic investments into the two types of offspring are subsequently converted into a number of offspring by dividing them by the energy costs to produce a single offspring. For small- and large-sized offspring these costs are proportional to $0.7 \cdot V_b$ and $1.3 \cdot V_b$, respectively. For mothers born with a small size-at-birth this implies that the biased energetic investment in producing offspring with small sizes-at-birth is even more pronounced when considered in terms of number of offspring produced, whereas for mothers born with a large size-at-birth the bias is dampened by the conversion to number of offspring produced.

The size-at-birth of the offspring produced is therefore on average smaller in the model implemented in `KlanjscekDEB2.h`, compared to the model with a single state-at-birth, which is implemented in `KlanjscekDEB.h`. As a consequence, the number of offspring produced is larger, which is the most likely reason for the finding that the population growth rate of the model with 2 states-at-birth is consistently larger than in case of a single state-at-birth (see the graphical output of the demo script `KlanjscekDEB`).

Code block 9.1.1.1.C

```
/*
 *============================================================================
 * Specify the fecundity of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * The number of offspring produced has to be specified for every possible
 * state at birth in the variable 'fecundity[][]'. The first index of this
 * variable refers to the number of the structured population, the second
 * index refers to the number of the birth state.
 *
```

```
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 *=====================================================================
 */

void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double *fecundity[POPULATION_NR])
{
  double        Er;

  if (lifestage[0] == 1)               // Only for adults
    {
      Er = (1-KAPPA)*EM*FOOD*G*(NU*pow(VOLUME, 2.0/3.0) + M*VOLUME)/(FOOD + G);
      fecundity[0][0]  = fecundity[0][1]  = max(Er - (1-KAPPA)*EM*M*G*VP,0);
      if (BirthStateNr == 0)
        {
          fecundity[0][0] *= (2.0/3.0)*KAPPA_R/(EM*(KAPPA*G + FOOD)*0.7*VB);
          fecundity[0][1] *= (1.0/3.0)*KAPPA_R/(EM*(KAPPA*G + FOOD)*1.3*VB);
        }
      else
        {
          fecundity[0][0] *= (1.0/3.0)*KAPPA_R/(EM*(KAPPA*G + FOOD)*0.7*VB);
          fecundity[0][1] *= (2.0/3.0)*KAPPA_R/(EM*(KAPPA*G + FOOD)*1.3*VB);
        }
    }
  else
    {
      fecundity[0][0] = 0;
      fecundity[0][1] = 0;
    }

  return;
}
```

In case of multiple states at birth the structures in the output file containing the stable population states is more complex. Consider for example the computation with the file `KlanjscekDEB2.h` that is executed when running the demo script `KlanjscekDEB`:

Command box 9.1.1.1.A

```
> output2 <- PSPMdemo("KlanjscekDEB2", c(0, 1.0, -0.02, 0.4, 1.0), clean=TRUE, force=TRUE)

Building executable KlanjscekDEB2demo.so ...

<...compilation output lines suppressed in this box...>

  1.00000000E+00  6.95508116E-01
  9.80000000E-01  6.82539595E-01
  9.60000000E-01  6.69183490E-01
<...output lines suppressed in this box...>
  4.40000000E-01  9.73367354E-02
  4.20000000E-01  5.99646757E-02
  4.00000000E-01  2.03431517E-02

#
# Executing : PSPMdemo("KlanjscekDEB2", c(0, 1, -0.02, 0.4, 1), NULL, NULL)
#
# Parameter values  :
#
#   Food      :  1             Kappa     :  0.8           Kappa_R   :  0.001
#   Nu        :  0.075         m         :  0.583         g         :  1.286
#   Vb        :  1E-09         Vp        :  1.73E-06      [Em]      :  0.7
#   ha        :  0.15
```

```
#
# Index of bifurcation parameter #1                          : 0
#
#  1:Food  2:PGR[ 0]  3:Tc[ 0]  4:S[ 0][ 0]  5:S[ 0][ 1]  6:S[ 0][ 2]  ..  12:S[ 0][ 8]  13:S[ 0][ 9]
#
```

Obviously, this model contains more parameters and hence there are many more columns in the output representing sensitivities of the population growth rate with respect to model parameters. Loading the first structure from the output file `KlanjscekDEB2-PGR-0000.csb` containing the stable population states and displaying its contents reveals the additional elements due to the multiple states at births:

Command box 9.1.1.1.B

```
> csbread("KlanjscekDEB2-PGR-0000.csb", 1)
$BifPars
[1] 1

$Parameters
 [1] 1.000e+00 8.000e-01 1.000e-03 7.500e-02 5.830e-01 1.286e+00 1.000e-09 1.730e-06 7.000e-01 1.500e-01

$PGR
[1] 0.6955081

$Pop00_StableBirthDist
      Bstate00  Bstate01
[1,] 0.7097291 0.2902709

$Pop00_BirthStates
     Istate00 Istate01 Istate02 Istate03
[1,]        0  7.0e-10        0        0
[2,]        0  1.3e-09        0        0

$Pop00_Bstate00
         StableDist   Istate00     Istate01     Istate02   Istate03  ReproVal
  [1,] 1.000000e+00  0.0000000 7.000000e-10 0.000000e+00 0.00000000  1.069154
  [2,] 9.258141e-01  0.1101390 8.928356e-09 7.625625e-09 0.01058176  1.154825
  [3,] 8.558545e-01  0.2202781 3.423292e-08 3.154797e-08 0.02537620  1.249224
<...output lines suppressed in this box...>
 [98,] 2.186738e-09 10.6834855 3.313724e-04 9.494863e-04 2.86850014  5.715571
 [99,] 1.472939e-09 10.7936245 3.367025e-04 9.735928e-04 2.91605415  3.380034
[100,] 1.000000e-09 10.9001686 3.418510e-04 9.971985e-04 2.96246948  0.000000

$Pop00_Bstate01
         StableDist   Istate00     Istate01     Istate02    Istate03  ReproVal
  [1,] 1.000000e+00  0.0000000 1.300000e-09 0.000000e+00 0.000000000 0.8309155
  [2,] 9.258768e-01  0.1101019 1.177775e-08 9.742355e-09 0.009863215 0.8974364
  [3,] 8.560174e-01  0.2202038 4.088023e-08 3.735042e-08 0.024398777 0.9706759
<...output lines suppressed in this box...>
 [98,] 2.187430e-09 10.6798867 3.321082e-04 9.527934e-04 2.871120863 4.3466694
 [99,] 1.473172e-09 10.7899886 3.374354e-04 9.769332e-04 2.918718544 2.5706244
[100,] 1.000000e-09 10.8964976 3.425811e-04 1.000571e-03 2.965176460 0.0000000
```

The first additional element of the list representing the equilibrium population state is `Pop00_StableBirthDist`, which specifies the stable distribution of offspring produced with the 2 possible states at birth that are defined in the model. Each of the rows of the element `Pop00_BirthStates` of the structure specifies a different state at birth with its columns specifying the value of the 4 individual state variables in that particular state. For each state at birth, a stable population distribution for individuals born in that particular state is stored in two-dimensional arrays, called `Pop00_Bstate00` and `Pop00_Bstate01` respectively. As before, these two-dimensional arrays contain as the first and last column the stable population density and the reproductive value, respectively, while the intervening columns contain the values of the individual state variables.

9.1.1.2 Periodic environments

The file `Medfly_periodic.h` implements a variant of the Medfly model that is discussed in section 3.2, in which juvenile medflies are periodically exposed to a very high mortality rate that decays exponentially within a short time period. Such a scenario could, for example, reflect a periodic treatment of the population with an insecticide that affects all juvenile individuals equally, irrespective of their age. This model is discussed in detail in De Roos (2008, see pp. 8-10) and will thus not be presented further here. The script `Medfly_periodic` that is included with the package as a demo can be used to obtain the results that are also shown in Figure 3 of De Roos (2008). Notice, however, that this computation takes a while to finish because the periodicity in the juvenile mortality makes it computationally very intensive.

The model implemented in the file `Medfly_periodic.h` illustrates that it is possible to carry out demographic analysis, i.e. calculation of the population growth rate as a function of a parameter and the sensitivity of the growth rate with respect to all model parameters, even in case of periodic environments. This does, however, not extend to equilibrium and evolutionary analysis, which are based on the assumption that the environment is in a constant, equilibrium state.

9.1.2 Equilibrium and evolutionary analysis with multiple states-at-birth

The two files `PNAS2002_5bs.h` and `Indet_growth_5bs.h` implement versions of the models implemented in the files `PNAS2002.h` and `Indet_growth.h` and discussed in sections 4.1-4.4 and 6.1-6.3, respectively, but with 5 states-at-birth instead of the unique state-at-birth accounted for in the original models. The analysis of these model versions with multiple states-at-birth is largely similar to the analysis of the original models and will hence not be discussed further. The scripts `deRoosPersson5` and `Indet_growth5` that are included with the package as demos carry out the same analysis steps as presented in detail in sections 4.4 and 6.3, respectively, but for the model versions with 5 states-at-birth. Instead, in the following I will only discuss for the model implemented in `PNAS2002_5bs.h` the details, in which this implementation differs from the original model implemented in `PNAS2002.h`.

The following code box defines two macros, `BIRTHSTATES` and `BIRTHSPREAD`, which determine the number of different states-at-birth and the variation in size-at-birth between the smallest and the largest offspring body size. The number of states-at-birth is defined equal to 5 in the routine `SetBirthStates()`, as shown below:

Code block 9.1.2.A

```
/*
 *=============================================================================
 *      DEFINITION OF THE LIFE HISTORY MODELS FOLLOWS BELOW
 *=============================================================================
 * Specify the number of states at birth for the individuals in all structured
 * populations in the problem in the vector BirthStates[].
 *=============================================================================
 */

#define BIRTHSTATES     5
#define BIRTHSPREAD     2.0

void SetBirthStates(int BirthStates[POPULATION_NR], double E[])
{
  BirthStates[0] = BIRTHSTATES;

  return;
}
```

Subsequently, the values of the different states-at-birth is set in the routine `StateAtBirth()`, dependent on the index `BirthStateNr` of the state-at-birth. The 5 states-at-birth in the model form

a set $\{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5\}$. The code box below shows that the length-at-birth in these 5 different states equals $\ell_b - \Delta/2$, $\ell_b - \Delta/4$, ℓ_b, $\ell_b + \Delta/4$ and $\ell_b + \Delta/2$, respectively, where Δ is the difference in size-at-birth between the smallest and the largest offspring as given by the macro `BIRTHSPREAD` (Remember that indices in `C` start at 0 and that the values adopted by `BirthStateNr` hence run from 0 to 4). Of course, for all states-at-birth the age of the individual is set to 0.

Code block 9.1.2.B

```
/*
 *======================================================================
 * Specify all the possible states at birth for all individuals in all
 * structured populations in the problem. BirthStateNr represents the index of
 * the state of birth to be specified. Each state at birth should be a single,
 * constant value for each i-state variable.
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 *======================================================================
 */

void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
{
  AGE = 0.0;
  LENGTH = LB + (((double)BirthStateNr)/((double)(BIRTHSTATES-1)) - 0.5)*BIRTHSPREAD;

  return;
}
```

The final routine that differs between the models with a unique state-at-birth and 5 states-at-birth is the routine specifying the fecundity of an individual, as this routine should not only specify the number of offspring produced, but also the state-at-birth of the offspring produced. The code box below shows that the model implemented in the file `PNAS2002_5bs.h` assumes that the distribution of the produced offspring is independent of the size-at-birth of the mother, since the variables `fecundity[0][0]` to `fecundity[0][4]` are assigned the same values irrespective of the index `BirthStateNr` or the state-at-birth `birthstate[][]`. All mothers produce 50% of their offspring with a length-at-birth equal to ℓ_b, 20% of their offspring each with a length-at-birth equal to $\ell_b - \Delta/4$ and $\ell_b + \Delta/4$ and 10% each with the most extreme lengths-at-birth of $\ell_b - \Delta/2$ and $\ell_b + \Delta/2$.

The demo script `deRoosPersson5` performs the same analysis steps for the PNAS2002 model with 5 states-at-birth, as carried out by the demo script `deRoosPersson` for the original model. Executing this script shows that there are at most quantitative differences, if at all, between the results of the two models. A similar finding is obtained when comparing the results of the demo script `Indet_growth5` that performs the analysis of the model presented in section 6.2 but with 5 states-at-birth with the results of the original model, the analysis of which was discussed in section 6.3. In both cases the additional states-at-birth hence hardly affect model predictions.

Code block 9.1.2.C

```
/*
 *======================================================================
 * Specify the fecundity of individuals as a function of the i-state
 * variables and the individual's state at birth for all populations in every
 * life stage.
 *
 * The number of offspring produced has to be specified for every possible
 * state at birth in the variable 'fecundity[][]'. The first index of this
 * variable refers to the number of the structured population, the second
 * index refers to the number of the birth state.
```

```
 *
 * Notice that the first index of the variable 'istate[][]' refers to the
 * number of the structured population, the second index refers to the
 * number of the individual state variable. The interpretation of the latter
 * is up to the user.
 *============================================================================
 */

void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double *fecundity[POPULATION_NR])
{
  double            fec;

  fecundity[0][0] = 0.0;
  fecundity[0][1] = 0.0;
  fecundity[0][2] = 0.0;
  fecundity[0][3] = 0.0;
  fecundity[0][4] = 0.0;
  if (lifestage[0] == 2)
    {
      fec = RM*R/(R + RH)*LENGTH*LENGTH;
      fecundity[0][0] = 0.1*fec;
      fecundity[0][1] = 0.2*fec;
      fecundity[0][2] = 0.5*fec;
      fecundity[0][3] = 0.2*fec;
      fecundity[0][4] = 0.1*fec;
    }

  return;
}
```

9.1.3 Other applications of multiple states-at-birth

The models with multiple states-at-birth discussed here only represent the basic type of application of this modeling feature. The option to account for multiple states-at-birth allows, however, for modeling a variety of scenarios. It goes too far to present this range of scenarios in detail and I will hence limit myself to pointing out a few examples.

As one example, multiple states-at-birth can be used to distinguish between the sexes in a population model. Two states-at-birth can then be defined, representing the male and female sex of an individual. An individual's sex can influence its life history through for example development and mortality. If in addition the fecundity of the (female) individuals is modeled following a particular type of mating structure, it might be necessary in to define the total number of mature males and/or females in the population as environment variables.

As another example, multiple states-at-birth make it possible to account for population-genetic processes in a model. For example, 3 states-at-birth could be used to model the 2 homozygous and the single heterozygous genotypes in a one locus-two allele population model. Multiple alleles would be possible to account for as well at the expenses of defining more states-at-birth. In this manner, the interplay between population-genetic processes and complex individual life histories could be analyzed for its population and even community consequences.

9.2 Pulsed reproduction

All previous chapters listed as one of the basic assumptions for the class of structured population models that can be analysed with this software package that reproduction is modeled with a function $\beta(\boldsymbol{\chi}, \boldsymbol{\chi}_b, E)$, representing the rate of offspring production, dependent on the individual state, the individual's state-at-birth and possibly on its environment. Reproduction is hence considered a

continuous process. If reproduction would occur as a pulsed process in time, the density of individuals in a population would change instantaneously as would its impact on its environment. This precludes that the environment is constant in time, which is a crucial assumption for the equilibrium and evolutionary analysis of structured population models. Demographic analysis, however, is still possible even when reproduction occurs as a pulsed process in time.

To model reproduction as a pulsed process in time in case of demographic analysis of a structured population, the time interval between successive reproduction events has to be defined for a model implemented in `C` using the macro constant `REPRODUCTION_INTERVAL`, as for example shown in the command box below.

Code block 9.2.A

```
// The following definition will force the program to consider reproduction pulses
#define REPRODUCTION_INTERVAL      1.0
```

Reproduction will be assumed a pulsed event whenever `REPRODUCTION_INTERVAL` is defined. Notice that it is not possible to have irregular intervals between reproductive pulses, the interval is necessarily constant and equal to the value to which `REPRODUCTION_INTERVAL` is set.

The model file `KlanjscekDEBpulsed.h`, which can be listed using the command `showpapm("KlanjscekDEBpulsed")` provides an example of a model that describes reproduction as a pulsed process (the model is also discussed in De Roos (2008)). The model implemented in this file is similar to the model implemented in the file `KlanjscekDEB2.h`, which is discussed in section 9.1.1, except for the fact that reproduction occurs as a pulsed event at regular time intervals of 1 time unit and all newborn individuals have the same state at birth.

To model the pulsed reproduction process an additional state variable characterizing an individual is introduced in the model, which represents the number of eggs that an adult individual has accumulated in its body. This content of the egg buffer is the 5th individual state variable in the model as shown in the following code box:

Code block 9.2.B

```
#define EGGS                istate[0][4]
```

The routine `Development` now contains additional statements specifying the dynamics for this individual state variable, which hence describe how the egg buffer is filling up in between two reproduction events. Naturally, this only occurs when an individual has matured, as shown in the code box below:

Code block 9.2.C

```
void Development(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                 double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                 double development[POPULATION_NR][I_STATE_DIM])
{
  double               dVda, dQda, dHda;
  double               Er;

  //  Assume growth always occurs
  dVda = max((FOOD*NU*pow(VOLUME, 2.0/3.0) - M*G*VOLUME)/(FOOD + G), 0);
  dQda = G*EM*(dVda + M*VOLUME);
  dHda = HA*Q/VOLUME;

  development[0][0] = 1.0;
  development[0][1] = dVda;                          // dV/da
  development[0][2] = dQda;                          // dQ/da
  development[0][3] = dHda;                          // dH/da
```

```
  if (lifestage[0] == 1)                                // Only for adults
    {
      Er = (1-KAPPA)*EM*FOOD*G*(NU*pow(VOLUME, 2.0/3.0) + M*VOLUME)/(FOOD + G);
      development[0][4]  = max(Er - (1-KAPPA)*EM*M*G*VP,0);
      development[0][4] /= EM*(KAPPA*G + FOOD)*VB/KAPPA_R;
    }
  else
    development[0][4] = 0;

  return;
}
```

Finally, if reproduction is modeled as a pulsed process the routine `Fecundity` has to specify the number of offspring produced at a reproduction event. As opposed to the case with continuous reproduction the fecundity is not a rate, but rather a number of offspring. As shown below, for the model implemented in `KlanjscekDEBpulsed.h` the fecundity is defined equal to the number of accumulated eggs. At the same time the egg buffer is emptied, i.e. `EGGS` set equal to 0. Hence, in case of pulsed reproduction, the routine `Fecundity` should not only define how many offspring are produced (possibly with different states at birth), but also how the individual state of the parent is changed when it reproduces.

Code block 9.2.D

```
void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
               double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
               double *fecundity[POPULATION_NR])
{
  if (lifestage[0] == 1)                // Only for adults
    {
      fecundity[0][0] = EGGS;
    }
  else
    fecundity[0][0] = 0;

  EGGS = 0.0;                        // Empty the egg buffer

  return;
}
```

The demo script `KlanjscekDEB` that is included with the package illustrates the analysis of the model in `KlanjscekDEBpulsed.h` at the same time as it analyses the related models implemented in `KlanjscekDEB.h` and `KlanjscekDEB2.h`.

9.3 Optional numerical settings

The values of the following options, modifying the numerical program settings, can be changed by means of `#define` statements in the model-specific file as illustrated in the code blocks 3.3.1.2 and 4.3.1.2 for models implemented in `R` and in code blocks 3.3.2.1 and 4.3.2.1 for models implemented in `C`.

Setting name	Default value	Interpretation
`MIN_SURVIVAL`	10^{-9}	Minimum survival probability at which an individual is considered dead
`MAX_AGE`	10^{6}	Absolute maximum age after which an individual is considered dead

Setting name	Default value	Interpretation
`DYTOL`	10^{-7}	Variable tolerance. The Newton iteration has converged when the norm of the right-hand side of the equations is less than `RHSTOL` *and* the norm of the consecutive adjustments to the solution vector of unknowns is less than `DYTOL`
`RHSTOL`	10^{-8}	Right-hand side tolerance. The Newton iteration has converged when the norm of the right-hand side of the equations is less than `RHSTOL` *and* the norm of the consecutive adjustments to the solution vector of unknowns is less than `DYTOL`
`ALLOWNEGATIVE`	0	If equal to 1 negative solution values are permissible, otherwise the program stops when a component of the solution vector becomes negative
`FULLSTATEOUTPUT`	2	If equal to 0 no output of the complete population state is produced. If equal to 1, output of the population state is produced in a binary file with `.csb` extension, with individuals originating from different states-at-birth weighted according to the stable distribution of produced offspring over states-at-birth and lumped into cohorts. If equal to 2, output of the population state is produced and individuals originating from different states-at-birth are stored as separate subpopulations.
`COHORT_NR`	100	Sets the number of cohorts in the output of the population state
`ODESOLVE_INIT_STEP`	0.1	Initial step size in the numerical integration of the ODEs. Initializes the globally accesible variable `Odesolve_Init_Step`.
`ODESOLVE_MIN_STEP`	10^{-8}	Smallest possible step size in the numerical integration of the ODEs. Initializes the globally accesible variable `Odesolve_Min_Step`.
`ODESOLVE_MAX_STEP`	10.0	Largest possible step size in the numerical integration of the ODEs. Initializes the globally accesible variable `Odesolve_Max_Step`.
`ODESOLVE_FIXED_STEP`	-	If defined, determines a value Δt, which forces the ODE integration method to include all time values $t = n\Delta t$ with $n = 0, 1, \ldots$ among its integration time steps in addition to possibly intervening time values enforced by the adaptive step size mechanism. Initializes the globally accesible variable `Odesolve_Fixed_Step`.
`ODESOLVE_ABS_ERR`	10^{-10}	Absolute error in the numerical integration of the ODEs. Initializes the globally accesible variable `Odesolve_Abs_Err`.
`ODESOLVE_REL_ERR`	10^{-8}	Relative error in the numerical integration of the ODEs. Initializes the globally accesible variable `Odesolve_Rel_Err`.

Setting name	Default value	Interpretation
`ODESOLVE_FUNC_TOL`	10^{-8}	Threshold value determining whether a stopping event in the numerical integration routine has been detected. Initializes the globally accesible variable `Odesolve_Func_Tol`.
`JACOBIAN_MIN_STEP`	10^{-7}	Absolute minimum change in variable when computing Jacobian matrix. Initializes the globally accesible variable `Jacobian_Min_Step`.
`JACOBIAN_STEP`	10^{-4}	Relative change in variable when computing Jacobian matrix. Initializes the globally accesible variable `Jacobian_Step`.
`JACOBIAN_UPDATES`	5	Number of Newton adjustments before the Jacobian matrix is computed anew. Initializes the globally accesible variable `Jacobian_Updates`.

Chapter 10

Analytical background

In this chapter I give a brief sketch of the computational approach, which is discussed in detail in Kirkilionis et al. (2001), Diekmann, Gyllenberg, and Metz (2003) and De Roos (2008). The description is far from complete, but only captures the basic idea of the computational machinery implemented in the package.

Consider the following generic model for the interaction of a size-structured consumer population foraging on an unstructured resource:

$$\frac{\partial n(t,s)}{\partial t} + \frac{\partial \left(g(s,R)n(t,s)\right)}{\partial s} = -\mu(s,R)\, n(t,s)$$

$$g(s_b,R)\, n(t,s_b) = \int_{s_b}^{s_m} \beta(s,R)\, n(t,s)\, ds$$

$$\frac{dR}{dt} = G(R) - \int_{s_b}^{s_m} \gamma(s,R)\, n(t,s)\, ds$$

In this model $n(t,s)$ represents the size distribution of the consumer population at time t and $R(t)$ is the resource density. The functions $g(s,R)$, $\beta(s,R)$ and $\mu(s,R)$ represent the growth rate in size of an individual with size s, its fecundity and its mortality rate, respectively. The function $G(R)$ described the autonomous dynamics of the resource R in the absence of consumers.

The computational approach is based on the idea that this model can also be expressed as a system of integro-differential equations of the following form:

$$\begin{aligned} b(t) &= \int_0^{\infty} \beta(s(t,a,R_t),R(t))\, \mathcal{F}(t,a,R_t)\, b(t-a)\, da \\ \frac{dR}{dt} &= G(R(t)) - \int_0^{\infty} \gamma(s(t,a,R_t),R(t))\, \mathcal{F}(t,a,R_t)\, b(t-a)\, da \end{aligned} \tag{10.1}$$

in which $b(t)$ is the population birth rate of the consumer population at time t and R_t represents the history of the resource density prior to time t, i.e. the function $R(\xi)$ with $\xi \in (-\infty, t]$.

The function $s(t,a,R_t)$ represents the body size of an individual consumer that is of age a at time t and has been exposed to the resource densities R_t since its birth. This body size is the integrated result of the growth rate $g(s,R)$ that the individual has experienced since birth:

$$s(t,a,R_t) = s^0 + \int_0^a g(s(t-\alpha,\alpha,R_{t-\alpha}),R(t-\alpha))\, d\alpha$$

The function $\mathcal{F}(t, a, R_t)$ represents the probability that an individual that is of age a at time t and has been exposed to the resource densities R_t since its birth is still alive. $\mathcal{F}(t, a, R_t)$ is related to the mortality rate $\mu(s, R)$ following:

$$\mathcal{F}(t, a, R_t) \;=\; \exp\left(-\int_0^a \mu(s(t-\alpha, \alpha, R_{t-\alpha}), R(t-\alpha))\, d\alpha\right)$$

Figure 10.1 below illustrates how the integro-differential equation system relates the birth rate in the past to the birth rate at time t through the intervening history of the resource density and the development of the consumers that have experienced this resource history.

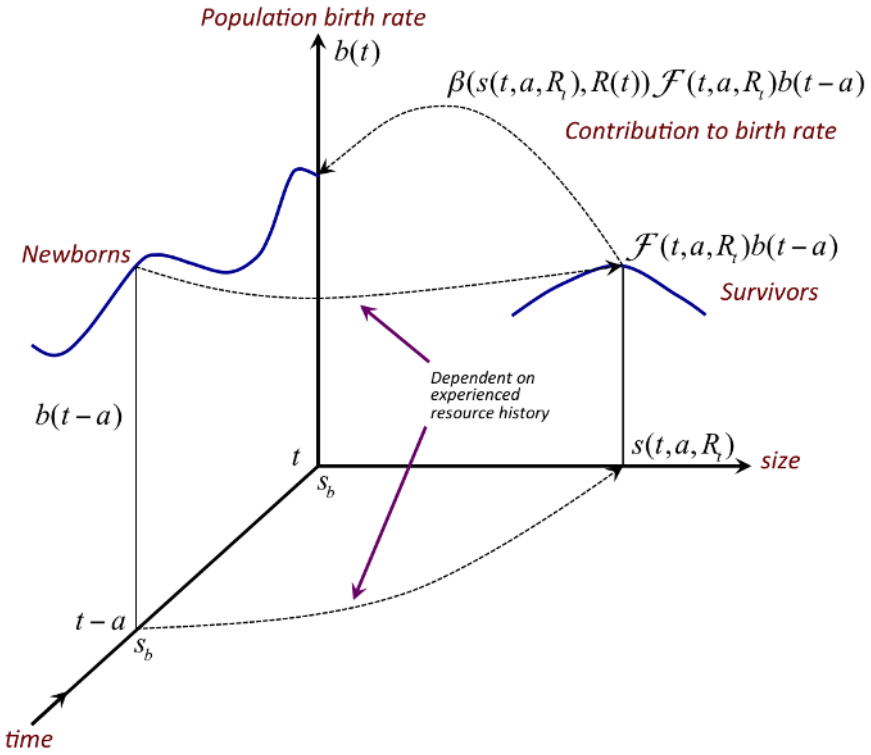

Figure 10.1: **Figure 10.1**: Schematic representation of the integro-differential equation system for the size-structured consumer-resource model, showing how the population birth rate at time $t - a$ contributes to the birth rate at time t through consumers of age a that have grown during their life from their size at birth s_b till their current body size $s(t, a, R_t)$ and have survived with a probability $\mathcal{F}(t, a, R_t)$, which both depend on the history of the resource R_t that these consumers have experienced.

10.1 The system of equations determining the population growth rate

For demographic analysis of a linear PSPM only the integral equation (10.1) is relevant. In linear PSPMs the individual life history is not influenced by any density dependence or by any dependence

on environment variables. We can hence drop the dependence of the development rate, fecundity and mortality rate on environment variables and generalize the integral equation (10.1) for an arbitrary choice of the individual state to:

$$b(t) \;=\; \int_0^\infty \beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a)\,b(t-a)\,da$$

in which $\boldsymbol{\chi}(a)$ is the state that individuals reach at age a provided they were born with state $\boldsymbol{\chi}_b$. $\boldsymbol{\chi}(a)$ is formally given by:

$$\boldsymbol{\chi}(a) \;=\; \boldsymbol{\chi}_b \;+\; \int_0^a g(\boldsymbol{\chi}(\alpha))\,d\alpha$$

and $\mathcal{F}(a)$ is the probability of survival up to age a:

$$\mathcal{F}(a) \;=\; \exp\left(-\int_0^a \mu(\boldsymbol{\chi}(\alpha))\,d\alpha\right)$$

Assuming exponential growth of the population birth rate:

$$b(t) = e^{ra}b(t-a)$$

leads to Lotka's integral equation for the population growth rate r:

$$\int_0^\infty e^{-ra}\beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a)\,da \;=\; 1 \tag{10.2}$$

Define the function $H(a,r)$ as the value of Lotka's integral up to age a:

$$H(a,r) \;=\; \int_0^a e^{-r\alpha}\beta(\boldsymbol{\chi}(\alpha))\,\mathcal{F}(\alpha)\,d\alpha$$

Equation (10.2) can then be expressed as:

$$H(\infty,r) \;=\; 1 \tag{10.3}$$

which is a non-linear equation for the population growth rate r. This is the equation that is solved by the software package for the unknown quantity r using an iterative approach based on the Newton-Chord method. For more details of the Newton-Chord method I refer to Kuznetsov (1995), which source I have used to a large extent for the iterative calculation of the solution $\tilde{r}$.

The central idea of the computational approach relates to the evaluation of the function $H(\infty,r)$, which is computed by solving an ordinary differential equation. To derive this ODE differentiate $\mathcal{F}(a)$ with respect to a using the chain rule:

$$\frac{d}{da}\mathcal{F}(a) \;=\; -\exp\left(-\int_0^a \mu(\boldsymbol{\chi}(\alpha))\,d\alpha\right)\frac{d}{da}\left(\int_0^a \mu(\boldsymbol{\chi}(\alpha))\,d\alpha\right)$$

Applying Leibniz rule for differentiation of an integral:

$$\frac{d}{d\theta}\left(\int_{a(\theta)}^{b(\theta)} f(\boldsymbol{\chi},\theta)d\boldsymbol{\chi}\right) = \int_{a(\theta)}^{b(\theta)} f_\theta(\boldsymbol{\chi},\theta)d\boldsymbol{\chi} + f(b(\theta),\theta)\,b'(\theta) - f(a(\theta),\theta)\,a'(\theta)$$

then leads to:

$$\frac{d\mathcal{F}}{da} \;=\; -\mu(\boldsymbol{\chi}(a))\,\mathcal{F}(a), \qquad\qquad \mathcal{F}(0) \;=\; 1$$

Similarly, differentiate $H(a,r)$ with respect to a and applying Leibniz rule yields:

$$\frac{dH}{da} \;=\; e^{-ra}\,\beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a), \qquad\qquad H(0) \;=\; 0$$

The value of $H(\infty,r)$ can hence be calculated by (numerical) integration of the ODEs:

$$\begin{cases} \dfrac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}), & \boldsymbol{\chi}(0) = \boldsymbol{\chi}_b \\ \dfrac{d\mathcal{F}}{da} = -\mu(\boldsymbol{\chi}(a))\,\mathcal{F}(a), & \mathcal{F}(0) = 1 \\ \dfrac{dH}{da} = e^{-ra}\,\beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a), & H(0) = 0 \end{cases}$$

In practice numerical integration of these ODEs is carried out up to $a = A_{max}$ with A_{max} either a fixed value or by $\mathcal{F}(A_{max}) = \epsilon$ with ϵ a very small value (i.e. $\epsilon = 10^{-9}$). Whenever an evaluation of the function $H(\infty,r)$ is required in the Newton iterations of equation (10.3) this system of ODEs has to be integrated numerically. Once, a solution $\tilde{r}$ has been found, the sensitivities of this solution with respect to the model parameters are calculated using numerical differentiation.

10.2 The system of equations determining an equilibrium

The idea discussed above for the demographic analysis of a linear PSPM extends to the computation of an equilibrium of a nonlinear PSPM. In such a nonlinear PSPM the fecundity and the development and mortality rates of individuals does depend on their environment, but in equilibrium this environment is necessarily constant: $\tilde{E}$. Therefore, Lotka's integral equation should determine as before the population growth rate r:

$$\int_0^\infty e^{-ra}\beta(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E})\,da \;=\; 1$$

Note that all parts of life history now depend on E. $\boldsymbol{\chi}$ and $\mathcal{F}$ do because of $g(\boldsymbol{\chi},E)$ and $\mu(\boldsymbol{\chi},E)$. However, r should equal 0 for equilibrium of the structured population:

$$\int_0^\infty \beta(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E})\,da \;=\; 1$$

In addition, the autonomous dynamics of the environment should be balanced by the impact of the population:

$$G(\tilde{E}) \;=\; \tilde{b}\int_0^\infty \gamma(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E})\,da$$

The survival rate $\mathcal{F}(a,\tilde{E})$ and the value of the cumulative reproduction integral:

$$H(a,\tilde{E}) \;=\; \int_0^a \beta(\boldsymbol{\chi}(\alpha,\tilde{E}),\tilde{E})\,\mathcal{F}(\alpha,\tilde{E})\,d\alpha$$

can be computed as before by solving the corresponding ODEs. To compute the impact of the population on the environment, define the function $I(a,\tilde{E})$ as:

$$I(a,\tilde{E}) \;=\; \int_0^a \gamma(\boldsymbol{\chi}(\alpha,\tilde{E}),\tilde{E})\,\mathcal{F}(\alpha,\tilde{E})\,d\alpha$$

$I(a,\tilde{E})$ represents the cumulative, expected impact that a single individual exerts on its environment until age a. Differentiating $I(a,\tilde{E})$ with respect to a yields after applying Leibniz rule:

$$\frac{dI}{da} \;=\; \gamma(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E}), \qquad\qquad I(0) \;=\; 0$$

The equilibrium of a nonlinear structured population model is therefore determined by the system of equations:

$$\begin{aligned} H(\infty,\tilde{E}) \;&=\; 1 \\ \tilde{b}I(\infty,\tilde{E}) \;&=\; G(\tilde{E}) \end{aligned}$$

which has to be solved *numerically* and *iteratively* for the unknowns $\tilde{E}$ and $\tilde{b}$. These equations are solved by the software package for the unknown quantities using the Newton-Chord method as discussed before. Whenever the functions $H(\infty,\tilde{E})$ and $I(\infty,\tilde{E})$ have to be evaluated in this iterative procedure, the following system of ODEs is integrated numerically:

$$\begin{cases} \dfrac{d\boldsymbol{\chi}}{da} = g(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E}), & \boldsymbol{\chi}(0,\tilde{E}) \;=\; \boldsymbol{\chi}_b \\ \dfrac{d\mathcal{F}}{da} = -\mu(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E}), & \mathcal{F}(0,\tilde{E}) \;=\; 1 \\ \dfrac{dH}{da} = \beta(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E}), & H(0,\tilde{E}) \;=\; 0 \\ \dfrac{dI}{da} = \gamma(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E}), & I(0,\tilde{E}) \;=\; 0 \end{cases}$$

10.3 Curve continuation and detection of bifurcation points

The package uses the Newton-Chord method with Broyden updating of the Jacobian matrix to solve for the root of the nonlinear system of equations that determines the population growth rate of linear PSPMs or the equilibrium of nonlinear PSPMs. In addition, pseudo-arclength continuation is used to compute a curve of either the population growth rate or the equilibrium as a function of

a single parameter. The numerical details about the Newton-Chord method as well as the pseudo-arclength continuation will not be discussed here. For details I refer to the appropriate sections in Kuznetsov (1995), which has been used as the basis for the implementations in the package. Both the Newton-Chord method as well as the pseudo-arclength continuation method make extensive use of partial derivatives of the system of equations with respect to variables and parameters. These partial derivatives, which for example make up the Jacobian matrix of the system of equations, are always computed numerically using a central-differencing approach.

The partial derivatives also play a role in the detection of bifurcation points, as explained in section 6.1. For example, the evolutionary analysis of PSPM using Adaptive Dynamics (AD) centers around the analysis of $s_x(y)$, which is the population growth rate of a mutant with trait y in an environment that is completely determined by a resident population with trait x (Geritz et al. 1998). An evolutionary fixed point occurs at $x = x^*$ where

$$\frac{\partial s_x(y)|_{x,y=x^*}}{\partial y} \;=\; 0$$

The evolutionary fixed point can be classified as a convergent stable strategy (CSS), an evolutionary repellor (ERP) or evolutionary branching point (EBP) based on the value of Geritz et al. (1998):

$$\frac{\partial^2 s_x(y)|_{x,y=x^*}}{\partial y^2} \qquad \text{and} \qquad \frac{\partial^2 s_x(y)|_{x,y=x^*}}{\partial x^2}$$

Because the equilibrium conditions for a structured model

$$\int_0^\infty \beta(\boldsymbol{\chi}(a,\tilde{E}),\tilde{E})\,\mathcal{F}(a,\tilde{E})\,da \;-\; 1 \;=\; R_0 \;-\; 1 = 0$$

is sign-equivalent with $s_x(y)$ AD analysis can be performed using R_0 and its (partial) derivatives with respect to resident and mutant traits x and y, respectively (Geritz et al. 1998). Hence, the detection of evolutionary fixed points, their classification as convergent stable strategies, repellors or branching points, as well as the continuation of these evolutionary singularities as a function of two parameters, which is discussed in sections 6.1 to 6.3 relies on the computation of these partial derivatives, which are computed numerically as pointed out above.

References

De Roos, A M. 1988. "Numerical methods for structured population models: The Escalator Boxcar Train." *Numerical Methods for Partial Differential Equations* 4: 173–95.

———. 1997. "A gentle introduction to physiologically structured population models." In *Structured Population Models in Marine, Terrestrial and Freshwater Systems*, edited by S Tuljapurkar and H Caswell, 119–204. Chapman-Hall, New York.

———. 2008. "Demographic analysis of continuous-time life-history models." *Ecology Letters* 11: 1–15.

De Roos, A M, and L Persson. 2002. "Size-dependent life-history traits promote catastrophic collapses of top predators." *Proceedings of the National Academy of Sciences* 99 (20): 12907–12.

———. 2013. *Population and community ecology of ontogenetic development.* Monographs in Population Biology 51. Princeton University Press, Princeton.

De Roos, A M, O Diekmann, and J A J Metz. 1992. "Studying the dynamics of structured population models: A versatile technique and its application to Daphnia." *American Naturalist* 139 (1): 123–47.

Dieckmann, U. 1997. "Can adaptive dynamics invade?" *Trends in Ecology & Evolution* 12 (4): 128–31.

Dieckmann, U, and R Law. 1996. "The dynamical theory of coevolution: A derivation from stochastic ecological processes." *Journal of Mathematical Biology* 34 (5-6): 579–612.

Diekmann, O, M Gyllenberg, and J A J Metz. 2003. "Steady-state analysis of structured population models." *Theoretical Population Biology* 63 (4): 309–38.

Geritz, S A H, E Kisdi, G Meszéna, and J A J Metz. 1998. "Evolutionarily singular strategies and the adaptive growth and branching of the evolutionary tree." *Evolutionary Ecology* 12 (1): 35–57.

Geritz, S A H, J A J Metz, and C Rueffler. 2016. "Mutual invadability near evolutionarily singular strategies for multivariate traits, with special reference to the strongly convergence stable case." *Journal of Mathematical Biology* 72 (4): 1081–99.

Kirkilionis, M A, O Diekmann, B Lisser, M Nool, B Sommeijer, and A M De Roos. 2001. "Numerical continuation of equilibria of physiologically structured population models. I. Theory." *Mathematical Models & Methods in Applied Sciences* 11 (6): 1101–27.

Kuznetsov, Y A. 1995. *Elements of applied bifurcation theory.* Heidelberg: Springer-Verlag.

Leimar, O. 2005. "The evolution of phenotypic polymorphism: Randomized strategies versus evolutionary branching." *American Naturalist* 165 (6): 669–81.

Metz, J A J, and O Diekmann. 1986. *The dynamics of physiologically structured populations.* Vol. 68. Lecture Notes in Biomathematics. Springer-Verlag, Heidelberg.

Metz, J A J, S A H Geritz, G Meszéna, F J A Jacobs, and J S van Heerwaarden. 1996. "Adaptive dynamics, a geometrical study of the consequences of nearly faithful reproduction." In *Stochastic and Spatial Structures of Dynamical Systems*, edited by S J van Strien and S M Verduyn-Lunel, 183–231. Amsterdam: KNAW Verhandelingen.

Zhang, L, U Dieckmann, and Å Brännström. 2017. "On the performance of four methods for the numerical solution of ecologically realistic size-structured population models." *Methods in Ecology and*

Evolution 8 (8): 948–56.