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 October 4, 2016

Contents

Demographic analysis of linear PSPMs

Equilibrium analysis of nonlinear PSPM

Evolutionary analysis of nonlinear PSPMs

Simulating the individual life history

Additional information

List of C code bloxes

List of Matlab and UNIX command boxes

Demographic analysis

Equilibrium analysis

Analysis of evolutionary fixed points

Simulating evolutionary dynamics

Simulating individual life histories in specific environments

UNIX command-line usage

Preface

This software package is distributed in the hope that it will be useful for 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 & 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 & Persson (2013). The approach is also shortly discussed in the Technical Appendices of that Monograph. 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, Matlab or Octave. The software can also be used from the command-line of any Unix-based system (Linux or Mac OS) without the overhead of R, Matlab or Octave. This manual focuses on the use of the software package from R or Matlab (its use from Octave should be the same).

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.

Enjoy!

André de Roos

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 & 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 four 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 characterises 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 parameterised curve of equilibrium states. Two types of special points can be detected in 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).

- **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 & 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 frontends that allows the software to be used from either R, Matlab or Octave. The implementation of the elements of the PSPM under study, however, have to be programmed in C, using the template files provided with the package. These design decisions have been inspired by preliminary tests of the computational efficiency. I would have preferred to use either R or Matlab instead of C for the implementation of the user-defined ingredients of the PSPM under study, but this would increase the execution time by roughly 2 orders of magnitude. Computations would then have been excruciatingly slow, which (hopefully) justifies the added difficulty of using C.

The basic methodology to numerically compute the equilibrium of a PSPM has been presented in Kirkilionis *et al.* (2001) and Diekmann *et al.* (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 18.

Short setup guide for Matlab

2.1 Prerequisites

When using Matlab the package uses the `mex` compiler to compile C files into a shared library that can be loaded for computations. The `mex` compiler needs a stand-alone C compiler to function properly. Newer versions of Mac OS X do not automatically include a compiler with the basic distribution and hence require installation of the command-line tools of Xcode, the development environment on Mac OS X. For freely available and supported compilers on Windows systems check the Matlab website (www.mathworks.com). In Matlab run the command `mex -setup` to select the C-compiler to use. Make sure that your setup of `mex` is working properly, consult the Matlab documentation for more information if necessary.

2.2 Package installation and testing

- Unpack the downloaded compressed file `PSPManalysis.zip`. This will create a directory `PSPManalysis`, which you can move to a location of your choice in your file system.

- Within Matlab change to the directory `PSPManalysis` that has been created by unpacking the compressed file `PSPManalysis.zip`.

- Add the directory `PSPManalysis` to your search path in Matlab.

- Change to the directory `Tests` and execute the Matlab script `alldemotests.m`, which subsequently calls the Matlab scripts `Medfly_demo.m` and `KlanjscekDEB_demo.m` and prints or plots the results in various ways. These scripts tests the working of the demographic analysis of the software package.

- Subsequently execute the Matlab script `allequitests.m`, which runs the scripts `PNAS2002equi_demo.m`, `Indet_growth_demo.m` and `KooijmanDEB_demo.m`. These scripts perform equilibrium computations of 3 different PSPMs and illustrate the detection of saddle-node and transcritical bifurcations in equilibrium curves and the continuation of these bifurcation points as a function of 2 model parameters (`PNAS2002equi_demo.m`). The script `Indet_growth_demo.m` illustrates the detection of an evolutionary singular point and its continuation as a function of a model parameter, the construction of the pairwise invasibility plot starting from the de-

tected evolutionary singular point, as well as the simulation of evolutionary dynamics in 2 life history parameters.

For demographic analysis of PSPMs use the problem file `Medly.h` as a template to implement the specific ingredients of the PSPM that you want to analyse, following the steps discussed in chapter 4. For equilibrium computations use the problem file `PNAS2002.h` as a template to implement the specific ingredients of the PSPM that you want to analyse, following the steps discussed in chapter 7.

2.2.1   Using the `mex` compiler under Matlab

2.2.2   Max OS X and Linux

When using Matlab the package needs the `mex` compiler to compile C files into a shared library that can be loaded for computations. The `mex` compiler requires a stand-alone C compiler to function properly. Newer versions of Mac OS X do not automatically include a compiler with the basic distribution and hence require installation of the command-line tools of XCode, the development environment on Mac OS X. In Matlab run the command `mex -setup` to select the C-compiler to use. Make sure that your setup of `mex` is working properly, consult the Matlab documentation for more information if necessary.

On Mac OS and Linux systems the `mex` compiler uses a shell script `mexopts.sh` to tailor the compilation settings to your system. The Matlab distribution includes a standard version of this shell script, but this has not worked very well for me. I therefore include here an adapted version of this shell script that works for me on Mac OS and Linux using Matlab 2012b. To use this version of the shell script it has to be placed in the directory where your Matlab version is looking for its configuration files. You can find out the name of this directory by issuing the command `prefdir` on the Matlab command line, as shown below:

```
2    >> prefdir

     ans =
4
     /Users/andre/.matlab/R2012b
6
```

If you put the `mexopts.sh` supplied here in that particular directory it will be used instead of the one supplied by default. The variables in the `mexopts.sh` that might have to be tuned for your system are the variables `CC`, `CFLAGS`, `LDFLAGS` and on Mac OS X possibly `CLIBS`.

The section in the `mexopts.sh` file supplied with the package that I have changed to get the package working properly on my Mac OS X (Yosemite) system is:

```
     #
2    # CLANG settings: Include -fopenmp as argument for threading support
     #
4    CC='/opt/local/bin/clang'
     CFLAGS="-fopenmp -fno-common -arch $ARCHS -isysroot $MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET"
6    LDFLAGS="-fopenmp -arch $ARCHS -Wl,-syslibroot,$MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET"
     #
8    # GCC settings
     #
10   # CC='/opt/local/bin/gcc'
     # CFLAGS="-fopenmp -fno-common -arch $ARCHS -isysroot $MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET"
12   # LDFLAGS="-fopenmp -arch $ARCHS -Wl,-syslibroot,$MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET"
     #
14   # Link with Matlab's libiomp5.dylib to prevent incompatibilities
```

```
     #
16   CLIBS="$MLIBS -L$TMW_ROOT/sys/os/$Arch -liomp5"
     #
18
```

In the above code, the variable `CC` is set to the compiler that I use on my Mac (I checked the package both with the `clang` and `gcc` compiler). I do not use the `clang` compiler that is standard supplied with `XCode` 7.2 on Mac OS X Yosemite, as this version of `clang` is old and does not support parallelisation of the program via `OpenMP`. The `clang` and `gcc` compiler I use I have installed from `macports`. Parallelisation speeds up the computations significantly in case individuals of a particular population can be born with different states at birth. Furthermore, to make use of the automatic parallelisation offered by `OpenMP` the command-line argument `-fopenmp` is added to both the `CFLAGS` and the `LDFLAGS` variable.

To get the package working properly on a Debian Linux system I have changed the variables `CC`, `CFLAGS` and `LDFLAGS` in the section for Linux systems to specify the compiler to use (I checked the package both with Intel's `icc` compiler and the `gcc` compiler) and have added the command-line argument for automatic parallelisation using `OpenMP` to the `CFLAGS` and `LDFLAGS` variables (options `-openmp` and `-fopenmp` for Intel's `icc` and the `gcc` compiler, respectively), as shown below:

```
     #
2    # ICC settings: Include -openmp as argument for threading support
     #
4    CC='icc'
     CFLAGS="-std=gnu99 -O3 -ipo -axAVX -msse3 -openmp -D_GNU_SOURCE"
6    LDFLAGS="-openmp -pthread -shared -Wl,--version-script,$TMW_ROOT/extern/lib/$Arch/$MAPFILE -Wl,--no-undefined"
     #
8    # GCC settings: Include -fopenmp as argument for threading support
     #
10   # CC='gcc'
     # CFLAGS='-D_GNU_SOURCE -fopenmp'
12   # LDFLAGS="-fopenmp -pthread -shared -Wl,--version-script,$TMW_ROOT/extern/lib/$Arch/$MAPFILE -Wl,--no-undefined"
     #
```

2.2.3 Windows

To compile `C` files into a shared library that can be loaded for computations using the `mex` compiler, check the Matlab website for freely available and supported compilers on Windows systems. In `Matlab` run the command `mex -setup` to select the C-compiler to use. Make sure that your setup of `mex` is working properly, consult the `Matlab` documentation for more information if necessary.

Demographic analysis of linear PSPMs

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

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

- 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*:

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

- 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*).

Implementation of an example model

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.

The implementation of this model, which I will refer to as the Medfly model, for analysis with the software package requires the specification of 10 pieces of C-code arranged in two different sections:

- Section 1: Problem dimensions, numerical settings and model parameters

- Section 2: 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 2 sections with 10 subsections. The code can be found in the file `Medfly.h` in the directory `Tests`. 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 in the directory Tests as a basis for the implementation, for example, by opening the file Medfly.h in Matlab's built-in editor and saving it with a different 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 characterised by a finite number of individual state variables. The number of state variables characterising 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 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).

⇒ | As listed in chapter 3, 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*:

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

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 chapter 14.

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 (see section 4.2.1).

4.1 Problem dimensions, numerical settings and model parameters

4.1.1 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 box 4.1: Definition of dimensions and numerical settings

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

6
     // Numerical settings: Optional (default values adopted otherwise)
8    #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
12   #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 characterises 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 chapter 17. 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 is in fact superfluous as they are defined equal to their default values (see chapter 17).

4.1.2 Definition of parameter names and values

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

Code box 4.2: Definition of parameter names and values

```
    // Descriptive names of parameters in parameter array (at least two parameters are required)
2   char  *parameternames[PARAMETER_NR] =
        { "Beta0", "Beta1", "AJ", "Mu0", "Mu1"};
4
    // Default values of all parameters
6   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).

4.1.3 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 box 4.3: Definition of aliases

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

4   // Aliases definitions for all parameters
    #define BETA0            parameter[ 0]   // Default: 47.0
6   #define BETA1            parameter[ 1]   // Default:  0.04
    #define AJ               parameter[ 2]   // Default: 11.0
8   #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 Matlab)! 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.

4.2 Definition of the individual life history

4.2.1 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 box 4.4: Specification of the number of possible states-at-birth

```
   /*
 2  * Specify the number of states at birth for the individuals in all structured
    * populations in the problem in the vector BirthStates[].
 4  */

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

10   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.

4.2.2   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 $\phi_j = \big(\phi_{j1}, \ldots, \phi_{jk}\big)$:

Code box 4.5: Specification of the value of all possible states-at-birth

```
   /*
 2  * Specify all the possible states at birth for all individuals in all
    * structured populations in the problem. BirthStateNr represents the index of
 4  * the state of birth to be specified. Each state at birth should be a single,
    * constant value for each i-state variable.
 6  *
    * Notice that the first index of the variable 'istate[][]' refers to the
 8  * 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.
    */
12
   void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
14 {
     AGE    = 0.0;
16
     return;
18 }
```

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 characterisation 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.

➠ 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 $\{\phi_1, \ldots, \phi_m\}$ 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.

4.2.3 Definition of boundaries between discrete stages

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

Code box 4.6: Definition of discrete stage boundaries

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

12  void IntervalLimit(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                       double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
14                     double limit[POPULATION_NR])
    {
16    if (lifestage[0] == 0)
        limit[0] = AGE - AJ;
18
      return;
20  }
```

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 4.2.4-4.2.7 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 4.2.4-4.2.7. 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 code box 4.6 also contains an array `E[]` as a variable. This array will contain the values of the environment variables during equilibrium computations of PSPMs (see chapters 6 to 8). 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 7.1 on page 39 for details). The appropriate value to use for the environment variables should be assigned

to the elements `E[e]` in the routine `StateAtBirth()` (see the next code box 4.5), after which it will keep the same value throughout all the subsequent routines.

4.2.4 Specification of continuous individual state development

Code box 4.7: Specification of continuous individual state development

```
   /*
 2  * 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
 4  * life stage.
    *
 6  * Notice that the first index of the variables 'istate[][]' and 'development[][]'
    * refers to the number of the structured population, the second index refers
 8  * to the number of the individual state variable. The interpretation of the
    * latter is up to the user.
10  */

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

18    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 4.2.3 concerning the dependence on the individual's state-at-birth.

4.2.5 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 box 4.8: Specification of discrete individual state changes

```
   /*
 2  * Specify the possible discrete changes (jumps) in the individual state
    * variables when ENTERING the stage specified by 'lifestage[]'.
 4  *
    * Notice that the first index of the variable 'istate[][]' refers to the
 6  * number of the structured population, the second index refers to the
    * number of the individual state variable. The interpretation of the latter
 8  * is up to the user.
    */

10
   void DiscreteChanges(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
12                     double *birthstate[POPULATION_NR], int BirthStateNr, double E[])
   {
14   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 4.2.3 concerning the dependence on the individual's state-at-birth.

4.2.6   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 box 4.9: Specification of fecundity

```
   /*
    * 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 4.2.3.

4.2.7 Specification of mortality

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

Code box 4.10: Specification of mortality

```
   /*
 2  * 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
 4  * life stage.
    *
 6  * Notice that the first index of the variable 'istate[][]' refers to the
    * number of the structured population, the second index refers to the
 8  * number of the individual state variable. The interpretation of the latter
    * is up to the user.
10  */

12  void Mortality(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
                   double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
14                 double mortality[POPULATION_NR])
    {
16    mortality[0] = MU0*exp(MU1*AGE);

18    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 4.2.3 concerning the dependence on the individual's state-at-birth.

Model analysis in Matlab

5.1 Executing of the `PSPMdemo` script

Once the model has been implemented, you can proceed carrying out its analysis, which in the simplest approach is performed by calling the Matlab-script `PSPMdemo.m` with the name of the file specifying the PSPM passed as a string argument. It is unnecessary to include the extension `.h` as part of the file name, the Matlab-script will strip the `.h` extension away if it is included. Therefore, the following two invocations of `PSPMdemo` are identical:

```
>> PSPMdemo('Medfly')
```

and

```
>> PSPMdemo('Medfly.h')
```

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

Command box 5.A: `PSPMdemo` call for a single parameter value

```
 >> PSPMdemo('Medfly', 'clean', 'force')

 Building executable Medflydemo.mexmaci64 ...

 #
 # Executing : PSPMdemo('Medfly', [], [], {})
 #
 # Parameter values  :
 #
 # Beta0    : 47           Beta1    : 0.04          AJ       : 11
 # Mu0      : 0.00095      Mu1      : 0.0581
 #
 #   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


 ans =

     []
```

The `PSPMdemo` function first compiles the model-specific file using the Matlab's `mex` compiler as is indicated by the first line of output of the script. This compilation step is only carried out when the executable (called `Medflydemo.mexmaci64` on my system) 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'` as in the Matlab 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 script 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, as set in the code box 4.2 on page 16.

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 code box 4.2. 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 script a vector of 5 elements of the following form:

$$[<index>\ <starting\ value>\ <step\ size>\ <minimum\ value>\ <maximum\ value>]$$

The first element indicates the index of the parameter in the array `parameter` (see code box 4.3) 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.

The following Matlab code illustrates this use of the `PSPMdemo` function for the Medfly model by computing the population growth rate as a function of `parameter[2]`, which is the value of A_j, 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 $11 \le A_j \le 20$.

Command box 5.B: PSPMdemo call to compute a curve over a parameter range

```
    >> [data desc] = PSPMdemo('Medfly', [2 11 0.1 11 20]);

    Executable Medflydemo.mexmaci64 is up-to-date

     1.10000000E+01  4.19056620E-01
     1.11000000E+01  4.15884247E-01
```

```
     1.12000000E+01  4.12762685E-01
 8   <...output lines suppressed in this box...>
     1.98000000E+01  2.53879994E-01
10   1.99000000E+01  2.52772251E-01
     2.00000000E+01  2.51674454E-01
12

14   #
     # Executing : PSPMdemo('Medfly', [2 11 0.1 11 20], [], {})
16   #
     # Parameter values  :
18   #
     #  Beta0    : 47            Beta1    : 0.04          AJ        : 11
20   #  Mu0      : 0.00095       Mu1      : 0.0581
     #
22   # Index of bifurcation parameter #1                      : 2
     #
24   #       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]
     #
26   >> data

28   data =

30          11     0.41906      13.167     0.0016159     -0.16459     -0.031982      -1.5264     -0.011325
           11.1     0.41588      13.282     0.0016019     -0.16429     -0.031467      -1.5323      -0.01148
32         11.2     0.41276      13.397     0.0015881     -0.16399     -0.030966      -1.5383     -0.011637
     <...output lines suppressed in this box...>
34         19.8     0.25388       23.15    0.00091906     -0.14471      -0.11128      -2.1725     -0.030825
           19.9     0.25277      23.262    0.00091464     -0.14453      -0.11027      -2.1815     -0.031129
36           20     0.25167      23.374    0.00091025     -0.14436      -0.10929      -2.1905     -0.031436
     >> sprintf(desc)
38
     ans =
40

42   #
     # Executing : PSPMdemo('Medfly', [2 11 0.1 11 20], [], {})
44   #
     # Parameter values  :
46   #
     #  Beta0    : 47            Beta1    : 0.04          AJ        : 11
48   #  Mu0      : 0.00095       Mu1      : 0.0581
     #
50   # Index of bifurcation parameter #1                      : 2
     #
52   #       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]
     #
```

Some of the intermediate lines of output generated by Matlab in this case are suppressed for brevity. When the PSPMdemo function is invoked in this way to compute parameter dependence, it generates two output variables, which in the Matlab command box shown above are assigned to the variables data and desc. The variable data consists of 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 data 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.

The output variable desc contains the description of the executed calculation, which is the textual information that is also printed to the Matlab console at the end of calculations. In fact, the PSPMdemo function prints its report on the calculations by execution of the statement sprintf(desc) as is also executed from the Matlab command-line in the box above.

5.2   Output files generated by the PSPMdemo script

The PSPMdemo function and module generates 2 output files when the script is only performing a single population growth rate calculation and 3 output files when the pop-

ulation 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` extension, `<NNNN>` is a 4-digit number that is unique for the current computation and `.<ext>` is the extension, which can be either `.err`, `.mat` or `.out`. Hence, the invocation of the `PSPMdemo` function for the Medfly model, as shown in Matlab command box 5.A, generates the output files `Medfly-PGR-0000.err` and `Medfly-PGR-0000.mat`, while the invocation of the `PSPMdemo` function for the Medfly model, as shown in Matlab command box 5.B, generates three output files: `Medfly-PGR-0001.err`, `Medfly-PGR-0001.mat` 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'`, as is the case in Matlab command box 5.A, 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>.mat` 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 contains a function `PSPMclean`, taking no arguments, to delete all `.bif`, `.err`, `.mat` 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>.mat` contains information on the stable population distribution for every parameter value for which the population growth rate is computed. For example, the file `Medfly-PGR-0001.mat` is generated by the invocation of the `PSPMdemo` function for the Medfly model shown in Matlab command box 5.B. By highlighting (selecting) this file in Matlab's "Current folder" window, its contents are displayed in Matlab's "File details" window. It consists of a series of population states, one for each of the parameter value for which the population growth rate has been computed, as shown in the following box (notice that some intermediate output is suppressed as was also the case in Matlab command box 5.B):

```
    State_1_100000E01 <1x1 struct>
2   State_1_110000E01 <1x1 struct>
    State_1_120000E01 <1x1 struct>
4   <...output lines suppressed in this box...>
    State_1_980000E01 <1x1 struct>
6   State_1_990000E01 <1x1 struct>
    State_2_000000E01 <1x1 struct>
```

The population state called `State_1_120000E01` pertains to the parameter value $A_j = 11.2$ as its name suggests. Loading this state into the Matlab workspace reveals its contents to be various arrays of numbers, as shown in the following box:

```
    >> load('Medfly-PGR-0001.mat', 'State_1_120000E01')
2   >> State_1_120000E01

4   State_1_120000E01 =
```

```
6                     BifPars: 11.2
                   Parameters: [47 0.04 11.2 0.00095 0.0581]
8                         PGR: 0.41276
          Pop00_BirthStates: 0
10                      Pop00: [100x3 double]
```

The first element (called `BifPars`) of the structure containing the population state `State_1_120000E01` is 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 structure `State_1_120000E01` 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 arrays 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 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 below:

```
   >> State_1_120000E01.Pop00
2
   ans =
4
            1            0            1
6       0.81299      0.50043         1.23
        0.66094       1.0009        1.513
8  <...output lines suppressed in this box...>
       1.5357e-09      48.542       8.4214
10      1.239e-09      49.042       4.6031
            1e-09      49.541            0
```

In the Medfly model individuals are only characterised by their age and hence their is only a single column with individual state variables. If individuals are characterised 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).

The last output file generated during the population growth rate has a name of the form `<Modelname>-PGR-<NNNN>.out`. This file contains the same information as is contained in the two output variables `data` and `desc` 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 `desc` and can be listed by the statement `sprintf(desc)` (see Matlab command box 5.B). Following this descriptive header the file contains columns with computational results that are also contained in the variable `data` (see Matlab command box 5.B), that is, the parameter values, population growth rates, generation times and sensitivities of the population growth rate to all model parameters. Matlab command box 5.B provides an example of the type of output generated by the computational module.

5.3   Required and optional arguments of `PSPMdemo`

As shown in Matlab command box 5.A 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` extension. All other arguments that can be passed to the `PSPMdemo` function are optional. A full listing of all required and optional arguments can be shown by calling up the help page of the `PSPMdemo` function using the command `help PSPMdemo`. The result is shown in Matlab command box 5.C.

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 Matlab command box 5.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 (`PARAMETER_NR`). When of this length the values will replace the default values of the parameters that are listed in the model specification file (see code box 4.2 for an example). If the array used for this third argument is not of the correct length `PARAMETER_NR`, it will simply be ignored.

The optional fourth argument of the `PSPMdemo` function is a Matlab cell array containing possible options that modify the behaviour of the computational module. A useful option is the `'test'` option, which can be passed to the computational module by using the cell array `{'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 cell array that modifies the behaviour of the computational module is the `'isort'` option, which can be passed to the computational module by using for example `{'isort', '1'}` as fourth argument to the `PSPMdemo` function. This option modifies the population state output that is stored in the output file with a name of the form `<Modelname>-PGR-<NNNN>.mat`. 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, see section 4.1.1) 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 Matlab where array indices start at 1). Therefore, passing `{'isort', '0'}` as option cell array to the `PSPMdemo` 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 `{'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 (see code box 4.2 in section 4.1.1 and chapter 17).

Three other, optional arguments can be passed to the `PSPMdemo` function: `'clean'`, `'force'` and `'debug'`. Unlike the previous arguments, which all modify the computations to be performed, these options modify the behaviour of the `PSPMdemo` function itself, in particular the compilation of the model specific file into a `mex` module that can be executed from Matlab. 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'`: This optional argument instructs the `PSPMdemo` function to delete all result files from previous calculations with the model, that is all files with names of the form `<Modelname>-PGR-<NNNN>.err`, `<Modelname>-PGR-<NNNN>.mat` and `<Modelname>-PGR-<NNNN>.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 contains a function `PSPMclean`, taking no arguments, to delete all `.err`, `.mat` and `.out` files and/or all executable files that are present in the current working directory.

- Option `'force'`: Using the option `'force'` instructs the `PSPMdemo` file to force re-compilation of the model specific file into a `mex` module that can be executed by Matlab. 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` extension is more recently changed than the compiled `mex` file. However, if for some unclear reason this automatic recompilation fails, the `'force'` option can be used to initiate re-compilation.

- Option `'debug'`: This option instructs the `PSPMdemo` function to turn on debugging flags while compiling the model specific file into a `mex` 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 Matlab 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.

Command box 5.C: `PSPMdemo` help page

```
    >> help PSPMdemo
2     PSPMdemo: Performs demographic analysis of a structured population model

4     Syntax:

6        [curvepoints, curvedesc] = ...
              PSPMdemo(modelname, curvepars, parameters, options, 'clean', 'force', 'debug')
8
      Arguments:
10
        modelname:  (string, required)
12                  Basename of the file with model specification. The file
                    should have extension '.h'. For example, the model 'Medfly'
14                  is specified in the file 'Medfly.h'

16      curvepars:  (row vector, optional, can be the empty vector [])
                    Vector of length 5, specifying:
18
                    curvepars(1): the index of the parameter to vary
20                  curvepars(2): the initial value of the parameter
                    curvepars(3): the step size in the parameter value
22                  curvepars(4): lower threshold, below which value of the
                                  parameter the computation stops
24                  curvepars(5): upper threshold, above which value of the
                                  parameter the computation stops
26
        parameters: (row vector, optional, can be the empty vector [])
28                  Vector of length PARAMETER_NR (set in the model program
                    file), specifying the values for the model parameters to
30                  use in the computation. Vectors of other lengths, including
                    an empty vector will be ignored.
32
        options:    (cell array, optional, can be the empty cell array {})
34                  Cell array with pairs of an option name and a value (for
                    example {'isort', '1'}) or single options (i.e. 'test').
36                  Possible option names and their values are:

38                  'isort',  '<index>': Index of i-state variable to use as
                                         ruling variable for sorting the
40                                       structured populations
                    'test'             : Perform only a single integration over
42                                       the life history, reporting dynamics
                                         of survival, R0 and i-state variables
44
        'clean':    (string. optional argument)
46                  Remove all the result files of the model before the
                    computation
48
        'force':    (string, optional argument)
50                  Force a rebuilding of the model before the computation

52      'debug':    (string, optional argument)
                    Compile the model in verbose mode and with debugging flag set
54
      Output:
56
        curvepoints: Matrix with output for all computed points along the curve
58
        curvedesc:   Column vector with strings, summarizing the numerical details
60                    of the computed curve (i.e., initial point, parameter values,
                      numerical settings used).
```

Equilibrium analysis of nonlinear PSPM

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 characterised 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 characterised 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*).

Implementation of an example model

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 & 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.

The implementation of this model for analysis with the software package requires the specification of 12 pieces of C-code arranged in three different sections:

- Section 1: Problem dimensions, numerical settings and model parameters

- Section 2: Definition of the individual life history functions, such as development (growth), fecundity and mortality.

- Section 3: 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 3 sections with 12 subsections. The code can be found in the file PNAS2002.h in the directory Tests. For ease of writing I will in the following sections often refer to this model as the PNAS model. The first 2 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 the previous chapter 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 characterised by a finite number of individual state variables. The number of state variables characterising 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).

⇛   As listed in chapter 6, 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*:

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

⟾ 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 chapter 14.

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 $\{\phi_1, \ldots, \phi_m\}$. The number m of possible states-at-birth is set dynamically in the model file (see section 7.2.1).

7.1 Problem dimensions, numerical settings and model parameters

7.1.1 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 box 7.1: Definition of dimensions and numerical settings

```
   // Dimension settings: Required
2  #define POPULATION_NR      1
   #define STAGES             3
4  #define I_STATE_DIM        2
   #define ENVIRON_DIM        3
6  #define INTERACT_DIM       4
   #define PARAMETER_NR       16
8
   // 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
12
   #define DYTOL              1.0E-7          // Variable tolerance
14 #define RHSTOL             1.0E-6          // Function tolerance
16 #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 chapter 17. 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 code box 7.1 (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 7.1) 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 chapter 17).

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 code box 7.1 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 8.5), one of which is a Matlab 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.

7.1.2 Definition of parameter names and values

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

Code box 7.2: Definition of parameter names and values

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

6    // Default values of all parameters
     double parameter[PARAMETER_NR] =
8        { 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 8.5). 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).

7.1.3 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 Matlab)! Similarly, the value of the individuals 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 box 7.3: Definition of aliases

```
     // Aliases definitions for all istate variables
2    #define AGE                 istate[0][0]
     #define LENGTH              istate[0][1]
4
     // Aliases definitions for all environment variables
6    #define R                   E[0]
     #define P                   E[1]
8    #define B                   E[2]

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

14   #define LB                  parameter[ 2] // Default: 7
     #define LV                  parameter[ 3] // Default: 27
16   #define LJ                  parameter[ 4] // Default: 110
     #define LM                  parameter[ 5] // Default: 300
18
     #define OMEGA               parameter[ 6] // Default: 9.0E-6
20
     #define IMAX                parameter[ 7] // Default: 1.0E-4
22   #define RH                  parameter[ 8] // Default: 1.5E-5

24   #define GAMMA               parameter[ 9] // Default: 0.006
     #define RM                  parameter[10] // Default: 0.003
26
     #define MUB                 parameter[11] // Default: 0.01
28
     #define A                   parameter[12] // Default: 5000.0
30   #define TH                  parameter[13] // Default: 0.1
     #define EPSILON             parameter[14] // Default: 0.5
32   #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 7.1 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 code box 7.3 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.

7.2   Definition of the individual life history

7.2.1   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 box 7.4: Specification of the number of possible states-at-birth

```
   /*
 2  * Specify the number of states at birth for the individuals in all structured
    * populations in the problem in the vector BirthStates[].
 4  */

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

10    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.

7.2.2   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 $\phi_j = \left(\phi_{j1}, \ldots, \phi_{jk}\right)$.

Code box 7.5: Specifying the value of all possible states-at-birth

```
   /*
 2  * Specify all the possible states at birth for all individuals in all
    * structured populations in the problem. BirthStateNr represents the index of
 4  * the state of birth to be specified. Each state at birth should be a single,
    * constant value for each i-state variable.
 6  *
    * Notice that the first index of the variable 'istate[][]' refers to the
 8  * 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.
    */
12
    void StateAtBirth(double *istate[POPULATION_NR], int BirthStateNr, double E[])
14  {
      AGE = 0.0;
16    LENGTH = LB;

18    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. Notice that the program does not automatically include individual age in its characterisation 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 $\{\phi_1, \ldots, \phi_m\}$ 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.

7.2.3 Definition of boundaries between discrete stages

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

Code box 7.6: Definition of discrete stage boundaries

```
   /*
    * 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])
   {
     switch (lifestage[0])
     {
       case 0:
         limit[0] = LENGTH - LV;
         break;
       case 1:
         limit[0] = LENGTH - LJ;
         break;
     }

     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 individuals 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 7.2.4-7.2.7 and 7.3.1 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 7.2.4-7.2.7 and 7.3.1. 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 ($\texttt{BirthStateNr} \geq \texttt{BirthStates}[p]$). 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 $\texttt{BirthStateNr} \geq \texttt{BirthStates}[p]$ that are inappropriate for the structured population with index p.

7.2.4   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 box 7.7: Specification of continuous individual state development

```
 /*
2  * 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
4  * life stage.
   *
6  * Notice that the first index of the variables 'istate[][]' and 'development[][]'
   * refers to the number of the structured population, the second index refers
8  * to the number of the individual state variable. The interpretation of the
   * latter is up to the user.
10 */

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

18
     return;
20 }
```

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 individuals 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 7.2.3 concerning the dependence on the individual's state-at-birth.

7.2.5   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 box 7.8: Specification of discrete individual state changes

```
   /*
2   * Specify the possible discrete changes (jumps) in the individual state
    * variables when ENTERING the stage specified by 'lifestage[]'.
4   *
    * Notice that the first index of the variable 'istate[][]' refers to the
6   * number of the structured population, the second index refers to the
    * number of the individual state variable. The interpretation of the latter
8   * is up to the user.
    */
10
   void DiscreteChanges(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
12                    double *birthstate[POPULATION_NR], int BirthStateNr, double E[])
   {
14   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 individuals 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 7.2.3 concerning the dependence on the individual's state-at-birth.

7.2.6 Specification of fecundity

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

Code box 7.9: Specification of fecundity

```
 1   /*
 2    * Specify the fecundity of individuals as a function of the i-state
 3    * variables and the individual's state at birth for all populations in every
 4    * life stage.
 5    *
 6    * The number of offspring produced has to be specified for every possible
 7    * state at birth in the variable 'fecundity[][]'. The first index of this
 8    * variable refers to the number of the structured population, the second
 9    * index refers to the number of the birth state.
10    *
11    * Notice that the first index of the variable 'istate[][]' refers to the
12    * number of the structured population, the second index refers to the
13    * number of the individual state variable. The interpretation of the latter
14    * is up to the user.
15    */
16
17   void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
18                  double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
19                  double *fecundity[POPULATION_NR])
20   {
21     fecundity[0][0] = 0.0;
22     if (lifestage[0] == 2)
23       fecundity[0][0] = RM*R/(R + RH)*LENGTH*LENGTH;
24
25     return;
26   }
```

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 chapter 7). 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 7.2.3.

7.2.7 Specification of mortality

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

Code box 7.10: Specification of mortality

```
   /*
2   * 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
4   * life stage.
    *
6   * Notice that the first index of the variable 'istate[][]' refers to the
    * number of the structured population, the second index refers to the
8   * number of the individual state variable. The interpretation of the latter
    * is up to the user.
10  */

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

20
      return;
22  }
```

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 $aP/(1+T_hB)$ as expressed by the function $\mu(\ell, P)$ in chapter 7.

> ⇒ Refer to the remarks in section 7.2.3 concerning the dependence on the individual's state-at-birth.

7.3   Feedback on and equilibrium condition of the environment

7.3.1   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 box 7.11: Specification of feedback impact on the environment

```
    /*
2    * For all the integrals (measures) that occur in interactions of the
     * structured populations with their environments and for all the integrals
4    * that should be computed for output purposes (e.g. total juvenile or adult
     * biomass), specify appropriate weighing function dependent on the i-state
6    * variables, the individual's state at birth, the environment variables and
     * the current life stage of the individuals. These weighing functions should
8    * be specified for all structured populations in the problem. The number of
     * weighing functions is the same for all of them.
10   *
     * Notice that the first index of the variables 'istate[][]' and 'impact[][]'
12   * refers to the number of the structured population, the second index of the
     * variable 'istate[][]' refers to the number of the individual state variable,
14   * while the second index of the variable 'impact[][]' refers to the number of
     * the interaction variable. The interpretation of these second indices is up
16   * to the user.
     */
18
   void Impact(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
20             double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
              double impact[POPULATION_NR][INTERACT_DIM])
22   {
     impact[0][0] = IMAX*R/(R + RH)*LENGTH*LENGTH;
24
     switch (lifestage[0])
26       {
       case 0:
28         impact[0][1] = OMEGA*LENGTH*LENGTH*LENGTH;
           impact[0][2] = 0;
30         impact[0][3] = 0;
           break;
32       case 1:
           impact[0][1] = 0;
34         impact[0][2] = OMEGA*LENGTH*LENGTH*LENGTH;
           impact[0][3] = 0;
36         break;
       case 2:
38         impact[0][1] = 0;
           impact[0][2] = 0;
40         impact[0][3] = OMEGA*LENGTH*LENGTH*LENGTH;
           break;
42       }

44     return;
   }
```

As explained in section 7.1.1 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 $h(\chi, \chi_b, E) = 1$), the total population biomass (when $h(\chi, \chi_b, E)$ equals the biomass of an individual with individual state χ and state-at-birth χ_b) or the total population birth rate in equilibrium (when $h(\chi, \chi_b, E)$ is the fecundity of an individual with individual state χ and state-at-birth χ_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 7.1.1 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.

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 7.11 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 in code box 7.11 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 $\{\phi_1, \ldots, \phi_m\}$ given by `birthstate[][]` and `BirthStateNr`, respectively) and the value of environment variables (`E[]`). In other words, the routine should only specify the weighing function $h(\chi, \chi_b, E)$. 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 7.2.3 concerning the dependence on the individual's state-at-birth.

7.3.2   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 chapter 6 environment variables can be of different types. As shown in the code box 7.12 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 statisfy $-\infty < -C \leq G(E, I) \leq C < \infty$ 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 (see code box 7.12 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 program (see code box 7.12 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 program (see code box 7.12 below) and its equilibrium condition has to be specified by identifying it with the appropriate feedback function I_i.

Code box 7.12: Specification of equilibrium condition of the environment

```
   /*
    * 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 feedbacks 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 code box 7.11. 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 χ_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:

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

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 code box 7.12 above, while its per capita growth rate $\epsilon aB/(1 + T_h B) - \delta$ 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 7.11.

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 code box 7.12 above. On line 36 of code box 7.12 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 code box 7.11).

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 *et al.*, 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[][]`.

Model analysis in Matlab

8.1 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 environ-

ment branching points can only be detected for environment variables that are of the type `PERCAPITARATE` (see section 7.3.2 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 characterised 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 18).

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 *et al.*, 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 *et al.*, 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.

8.2 Arguments of the `PSPMequi` script

Once the model has been implemented, you can proceed carrying out its analysis with the Matlab-script `PSPMequi.m`. The syntax for calling `PSPMequi` is shown in the Matlab command box below.

Command box 8.A: General syntax of a `PSPMequi` call

```
2    >> [curvepoints, curvedesc, bifpoints, biftypes] = ...
                   PSPMequi(modelname, biftype, startpoint, stepsize, parbnds, parameters, options,...
                            'clean', 'force', 'debug')
```

A call to `PSPMequi` requires 6 obligatory arguments and can in addition include 3 optional arguments. A short explanation of all arguments is provided by calling the help page of the Matlab-script `PSPMequi.m` with the command `help PSPMequi` (see the result in the Matlab command box 8.I).

The 7 obligatory arguments to the `PSPMequi` function are the following:

1. The first argument to the Matlab-script `PSPMequi` is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension `.h` as part of the file name, the Matlab-script will strip the `.h` extension away if it is included. The Matlab-command to analyse the PNAS model that will be used as illustrations below will therefore all take `'PNAS2002'` as their first argument.

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 8.1 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 Matlab (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:

   $$[<parameter> <environment\ variables> <population\ birth\ rates>]$$

   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:

   $$[<parameter> <environment\ variables> <population\ birth\ rates> <parameter\ 2>]$$

   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 Matlab (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:

[<*index 1*> <*minimum 1*> <*maximum 1*>]

The first element of the vector indicates the index of the parameter in the array parameter (see code box 7.3) 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:

[<*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 obligatory argument of the PSPMequi function is a Matlab (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (PARAMETER_NR). When of this length the values will replace the default values of the parameters that are listed in the model specification file (see code box 7.2 for an example). If the array used for this sixth argument is not of the correct length PARAMETER_NR, for example when an empty vector is passed as the sixth argument, it will simply be ignored.

7. The seventh and last obligatory argument of the PSPMequi function is a Matlab cell array containing possible options that modify the behaviour 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 Matlab where array indices start at 1).

- Option pair {'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 7.3.2 above).

- Option pair {'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.

- Option pair {'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.

- Option pair {'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 {'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.

- Option pair {'envZE', 'i'}: This option pair can be specified several times as part of the option cell array. 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 code box 7.12) 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 7.3.2 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.

- Option pair {'popZE', 'i'}: This option pair can be specified several times as part of the option cell array. 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).

- Option pair {'isort', 'i'}: This option modifies the output of the equilibrium state of the populations, which are stored in an output file with a name of the form <Modelname>-<Type>-<NNNN>.mat (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 {'isort', '0'} as option cell array 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 {'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 box 7.2 in section 7.1.1 as well as chapter 17).

- Option {'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.

- Option {'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.

- Option `{'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.

- Option `{'test'}`: The last possible option that can be passed to the `PSPMequi` function as part of the option cell array 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'`. Unlike the previous arguments, which all modify the computations to be performed, these options modify the behaviour of the `PSPMequi` function itself, in particular the compilation of the model specific file into a `mex` module that can be executed from Matlab. 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'`: This optional argument instructs the `PSPMequi` function to delete all result files that have been generated during previous calculations with the model, i.e. as a result of previous calls to `PSPMequi`. These result files have names of the form `<Modelname>-<Type>-<NNNN>.bif`, `<Modelname>-<Type>-<NNNN>.err`, `<Modelname>-<Type>-<NNNN>.mat` and `<Modelname>-<Type>-<NNNN>.out`, 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. either `BP`, `BPE`, `EQ`, `ESS`, `LP` or `PIP`, 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 contains a function `PSPMclean`, taking no arguments, to delete all `.bif`, `.err`, `.mat` and `.out` files and/or all executable files that are present in the current working directory.

- Option `'force'`: Using the option `'force'` instructs the `PSPMequi` file to force re-compilation of the model specific file into a `mex` module that can be executed by Matlab. 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 `mex` file. However, if for some unclear reason this automatic recompilation fails, the `'force'` option can be used to initiate re-compilation.

- Option `'debug'`: This option instructs the `PSPMequi` function to turn on debugging flags while compiling the model specific file into a `mex` 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 Matlab 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.

8.3 Output variables of the PSPMequi script

When calling the PSPMequi function it first compiles the model-specific file called `<Model>.h` using the Matlab's mex compiler into a compiled, computational module. This compilation step is only carried out when the executable (called `<Model>equi.mexmaci64` on my system) 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'` as discussed in the previous section. For example, for the PNAS model, the implementation of which is specified in the file PNAS2002.h, the executable is called PNAS2002equi.mexmaci64 (on other operating systems this file may be called PNAS2002equi.mex). Following successful compilation the PSPMequi function executes the compiled, computational module with the 7 arguments that are passed to the PSPMequi function as obligatory input (see the previous section).

The computational module generates up to 4 output variables on execution (see the help page on PSPMequi in Matlab command box 8.I). The first output variable curvepoints contains the numerical information of the points along the computed curve. This variable 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 code box 7.11), the per capita growth rate of all environment variables for which this is relevant (those of the type PERCAPITARATE, see section 7.3.2), 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 de option cell array) 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 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 chapters 9 to 11 for more details). These partial derivatives characterise 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 8.4 below).

The second output variable `curvedesc` (see Matlab command box 8.I), 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 Matlab console at the end of calculations. In fact, the `PSPMequi` function prints its report on the calculations by execution of the statement `sprintf(curvedesc)`. More details about the content of the description variable `curvedesc` is provided in the section below discussing the example of a model analysis using the `PSPMequi` function.

The third and fourth output variables are only non-empty for computations of equilibrium curves (type `'EQ'`). The third output variable `bifpoints` (see Matlab command box 8.I) contains the same type of information as the first output variable `curvepoints`, but now only for the detected bifurcation points along the computed curve. Finally, the fourth and last output variable `biftypes` (see Matlab command box 8.I) is a cell array with strings 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 cell array `biftypes` characterises the corresponding row in the output variable `bifpoints`.

8.4   An example session using the `PSPMequi` script

To illustrate the use of the `PSPMequi` function I will discuss the analysis of the PNAS model, presented in chapter 7. The statements below are taken from the Matlab-script `PNAS2002equi_demo.m`, which executes in addition to the statements represented here, Matlab commands to visualize the computed results in graphs. It is therefore recommended to run the `PNAS2002equi_demo.m` script at the same time as reading the explanation in this section.

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 characterised 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 options

{'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 equi-
librium 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 8.B 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 $4.0 \cdot 10^{-4}$.

The sixth argument in the call to the PSPMequi function shown below is an empty Matlab
row vector []. This argument represents the parameter values to be used for the com-
putation. Because it is an empty vector, it is ignored and the default parameter values,
as specified in code box 7.2, are used. This sixth argument will also be an empty row
vector in all further calls to the PSPMequi function discussed below and will therefore
from hereon be ignored.

Command box 8.B: Starting PSPM analysis from a trivial equilibrium

```
   >> [data1, desc1, bdata1, btype1] = PSPMequi('PNAS2002', 'EQ', [1.0E-06 1.0E-06], 0.5, [1 0 4E-4], [],...
2                                  {'popZE', '0', 'envZE', '1', 'envZE', '2'}, 'clean', 'force');

4    Building executable PNAS2002equi.mexmaci64 ...

6      1.00000000E-06  1.00000000E-06
       1.35355339E-06  1.35355339E-06
8      1.70710678E-06  1.70710678E-06
     <...output lines suppressed in this box...>
10     8.77817459E-06  8.77817459E-06
       8.85690312E-06  8.85690312E-06  ****  BP #0   ****
12     9.13172798E-06  9.13172798E-06
     <...output lines suppressed in this box...>
14     3.34047294E-04  3.34047294E-04
       3.69402633E-04  3.69402633E-04
16     4.04757972E-04  4.04757972E-04

18
     #
20   # Executing : PSPMequi('PNAS2002', 'EQ', [1E-06 1E-06], 0.5, [1 0 0.0004], [], {'popZE', '0', 'envZE', '1', 'envZE', '2'})
     #
22   # Parameter values  :
     #
24   #  Rho     :  0.1        Rmax    :  1E-06      Lb      :  7
     #  Lv      :  27         Lj      :  110        Lm      :  300
26   #  Beta    :  9E-06      Imax    :  0.0001     Rh      :  1.5E-05
     #  Gamma   :  0.006      Rm      :  0.003      Mub     :  0.01
28   #  A       :  5000       Th      :  0.1        Epsilon :  0.5
     #  Delta   :  0.01
30   #
     # Index of bifurcation parameter #1                       : 1
32   #
     #    1:Rmax    2:E[ 0]    3:E[ 1]    4:E[ 2]    5:b[ 0] 6:I[ 0][ 0] 7:I[ 0][ 1]  ..  10:pcgE[ 1] 11:R0[ 0] 12:RHS norm
34   #
     >> data1
36
     data1 =
38
         1e-06       1e-06         0         0         0         0         0  ..        -0.01         0         0
40    1.3536e-06  1.3536e-06        0         0         0         0         0  ..        -0.01         0  4.2352e-17
      1.7071e-06  1.7071e-06        0         0         0         0         0  ..        -0.01         0  4.2352e-17
42   <...output lines suppressed in this box...>
      0.00033405  0.00033405        0         0         0         0         0  ..        -0.01     4430.2  9.7578e-16
44     0.0003694   0.0003694        0         0         0         0         0  ..        -0.01     4491.5  1.0842e-15
      0.00040476  0.00040476        0         0         0         0         0  ..        -0.01     4542.8    1.03e-15
46
     >> bdata1
48
     bdata1 =
50
      8.8569e-06  8.8569e-06        0         0         0         0         0  ..        -0.01         1         0
```

```
52
     >> btype1
54
     btype1 =
56
         'BP #0'
58
```

In the output shown in the box above a large number of output lines have been suppressed and several intermediate columns have been deleted from the output, because the page width does not allow them to be shown completely. Consult the listing in your Matlab 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') and subsequently compiles the computational module using the mex compiler (notice the somewhat superfluous option 'force'). 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 box 7.2). 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 output variable of the script, here assigned to desc1, which is printed to the Matlab console using the statement sprintf(desc1). 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 box 7.2 on page 42. Furthermore, a header line is printed with a short description of each of the columns in the output matrix, which in the call to PSPMequi shown in the box above is assigned to the variable data1. As shown in the box above, executing the command data1 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 output variable, which is assigned to bdata1. Displaying bdata1 reveals that it has the same layout as the output matrix data1, 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 cell array btype1, 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 8.B, the PNAS2002equi_demo.m script also uses the output variables data1, bdata1 and btype1 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 the output variable bdata1. Starting from that point the call to PSPMequi shown in the next Matlab 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
{'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
bdata1 (although bdata1(5) is of course equal to 0).

Command box 8.C: Computation of the consumer-resource equilibrium

```
>> [data2, desc2, bdata2, btype2] = PSPMequi('PNAS2002','EQ',bdata1([1 2 5]),0.2,[1 0 4E-4],[],{'envZE','1','envZE','2'});

Executable PNAS2002equi.mexmaci64 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', [8.8569E-06 8.8569E-06 0], 0.2, [1 0 0.0004], [], {'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] 6:I[ 0][ 0] 7:I[ 0][ 1]  ..  10:pcgE[ 1] 11:R0[ 0] 12:RHS norm
#
>> data2

data2 =

  8.8569e-06  8.8569e-06         0         0         0         0         0  ..     -0.01        1  4.9364e-08
  9.0569e-06  8.8569e-06         0         0 1.9067e-09 1.9999e-08 3.2751e-09  .. -0.0099918        1  4.9364e-08
  9.2569e-06  8.8569e-06         0         0 3.8133e-09 3.9998e-08 6.5502e-09  .. -0.0099836        1  4.9364e-08
<...output lines suppressed in this box...>
  0.00038212  8.8569e-06         0         0 3.5586e-06 3.7326e-05 6.1127e-06  ..  0.0052351        1  5.0051e-08
   0.0003966  8.8569e-06         0         0 3.6966e-06 3.8774e-05 6.3497e-06  ..  0.0058241        1  5.0105e-08
  0.00041107  8.8569e-06         0         0 3.8346e-06 4.0222e-05 6.5868e-06  ..  0.0064129        1  5.0161e-08

>> bdata2

bdata2 =

   0.0002536  8.8569e-06         0         0 2.3333e-06 2.4475e-05  4.008e-06  .. -9.5387e-11        1  5.6287e-09

>> btype2

btype2 =

     'BPE #1'
```

As in the previous Matlab command box a large number of output lines and columns have
been suppressed to fit the page width of this manual. Please consult the listing in your
Matlab 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 8.C is similar to the output as discussed following Matlab command box 8.B. The
data of the computed equilibrium points are assigned to the variable data2. 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 returned as the third output variable, which is assigned to `bdata2`. Displaying `bdata2` reveals that it has the same layout as the output matrix `data2`, 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 corresponding element of the fourth output variable, assigned to the cell array `btype2`, contains the descriptive string `'BPE #1'` that is also printed to the console on detection of the branching point.

As before, the `PNAS2002equi_demo.m` script also plots the output variables `data2`, `bdata2` and `btype2` that result from the call to the `PSPMequi` function shown in box 8.C 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 `bdata2`. Starting from that point the call to `PSPMequi` shown in the next Matlab 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 with an empty cell array for the 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 `bdata2` 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 `bdata2`. Inspection of `bdata2(1:5)`, however, shows that both `bdata2(3)` and `bdata2(4)` are equal to 0:

```
>>  bdata2(1:5)

ans =

    0.0002536   8.8569e-06          0          0   2.3333e-06
```

Obviously, `bdata2(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 Matlab command box 8.C. 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 `bdata2(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 `{'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 `bdata2`) 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 box 7.11 and 7.12). Hence, the vector `bdata2([1 2 3 7 5])` is used as initial point for the computation shown in the next box:

Command box 8.D: Computing the predator-consumer-resource equilibrium

```
>> [data3, desc3, bdata3, btype3] = PSPMequi('PNAS2002', 'EQ', bdata2([1 2 3 7 5]), -0.1 , [1 0 4E-4], [], {});

Executable PNAS2002equi.mexmaci64 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', [0.000253602 8.8569e-06 0 4.00802e-06 2.33334e-06], -0.1, [1 0 0.0004], [], {})
#
# 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] 6:I[ 0][ 0] 7:I[ 0][ 1]  .. 10:pcgE[ 1] 11:R0[ 0] 12:RHS norm
#
>> data3

data3 =

   0.0002536  8.8569e-06           0  4.008e-06  2.3333e-06  2.4475e-05  4.008e-06  ..  -8.5897e-11       1  5.1658e-08
  0.00025258  8.8587e-06  9.5321e-08  4.008e-06  2.3617e-06  2.4372e-05  4.008e-06  ..  -6.9538e-11       1  3.1014e-08
  0.00025155  8.8605e-06  1.9056e-07  4.008e-06  2.3904e-06  2.4269e-05  4.008e-06  ..  -4.2689e-11       1  2.1533e-08
<...output lines suppressed in this box...>
  0.00038981  0.00023961  0.00013297  4.008e-06  0.00029263   1.502e-05  4.008e-06  ..  -5.8209e-10       1  2.3378e-07
  0.00039702  0.00024647  0.00013329  4.008e-06  0.00029352  1.5055e-05  4.008e-06  ..   -5.803e-10       1  2.3306e-07
  0.00040423  0.00025335   0.0001336  4.008e-06  0.00029438  1.5087e-05  4.008e-06  ..  -5.7618e-10       1  2.3141e-07

>> bdata3

bdata3 =

  8.4499e-05  1.3233e-05  3.8454e-05  4.008e-06  5.2566e-05  7.5266e-06  4.008e-06  ..  -1.6414e-10       1  6.6251e-08

>> btype3

btype3 =

    'LP'
```

In the output shown in the box above a large number of output lines have been suppressed and even more intermediate columns then in the previous Matlab output box have been deleted, because the page width does not allow them to be shown completely. Consult the listing in your Matlab 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 8.D is the same as in Matlab command box 8.C. The data of all computed equilibrium points making up the curve are assigned to the variable `data3` and the description of the computed curve is assigned to the variable `desc3`. Issuing the command `sprintf(desc3)` 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 stored in the output variable `bdata3`, whereas its description `'LP'` is stored in the first element of the cell array `btype3`. 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 `PNAS2002equi_demo.m` script uses `data3`, `bdata3`, and `btype3` 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 Matlab command box 8.B and 8.C).

The following 3 calls to `PSPMequi` that are executed by the `PNAS2002equi_demo.m` script, illustrated in Matlab command box 8.E, 8.F and 8.G, 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 script the vector `[1 0 4.0E-4 11 0 0.1]` indicating that the parameters with index 1 and 11 are to be varied (see code box 7.3) 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 `bdata1`, which was detected during the computation of the trivial equilibrium without any consumers and predators (see Matlab command box 8.B). 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 `[bdata1([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 Matlab command box 8.E hence equals the cell array `{'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 8.E: Computing the consumer invasion boundary

```
   >> [data4, desc4] = PSPMequi('PNAS2002', 'BP', [bdata1([1 2]) 0.01], 0.05 , [1 0 4.0E-4 11 0 0.1], [],...
2                           {'envZE', '1', 'envZE', '2', 'popBP', '0'});

4    Executable PNAS2002equi.mexmaci64 is up-to-date

6      8.85690312E-06  8.85690312E-06  1.00000000E-02
       8.88483087E-06  8.88483087E-06  1.03112655E-02
8      8.91358293E-06  8.91358293E-06  1.06066164E-02
```

```
     <...output lines suppressed in this box...>
10     3.94598924E-04  3.94598924E-04  8.27003296E-02
       3.98126241E-04  3.98126241E-04  8.27343996E-02
12     4.01653833E-04  4.01653833E-04  8.27678949E-02

14
     #
16   # Executing : PSPMequi('PNAS2002', 'BP', [8.8569E-06 8.8569E-06 0.01], 0.05, [1 0 0.0004 11 0 0.1], [],
                            {'envZE', '1', 'envZE', '2', 'popBP', '0'})
18   #
     # Parameter values  :
20   #
     #  Rho       :  0.1          Rmax      :  8.8569E-06   Lb        :  7
22   #  Lv        :  27           Lj        :  110          Lm        :  300
     #  Beta      :  9E-06        Imax      :  0.0001       Rh        :  1.5E-05
24   #  Gamma     :  0.006        Rm        :  0.003        Mub       :  0.01
     #  A         :  5000         Th        :  0.1          Epsilon   :  0.5
26   #  Delta     :  0.01
     #
28   # Index of bifurcation parameter #1                       : 1
     # Index of bifurcation parameter #2                       : 11
30   # Index of structured population with transcritical bifurcation: 0
     #
32   #    1:Rmax    2:E[ 0]    3:E[ 1]    4:E[ 2]    5:b[ 0]      6:Mub 7:I[ 0][ 0] 8:I[ 0][ 1] 9:I[ 0][ 2]  ..  13:RHS norm
     #
34   >> data4

36   data4 =

38     8.8569e-06  8.8569e-06         0          0          0       0.01         0          0          0  ..  4.9204e-08
       8.8848e-06  8.8848e-06         0          0          0   0.010311         0          0          0  ..  1.3274e-08
40     8.9136e-06  8.9136e-06         0          0          0   0.010607         0          0          0  ..  1.5157e-08
       <...output lines suppressed in this box...>
42     0.0003946   0.0003946         0          0          0     0.0827         0          0          0  ..  9.5602e-10
       0.00039813  0.00039813        0          0          0   0.082734         0          0          0  ..  9.6191e-10
44     0.00040165  0.00040165        0          0          0   0.082768         0          0          0  ..  9.6506e-10
```

(Once again, consult your Matlab 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).

As shown in the Matlab command box above the output of the PSPMequi function is stored in only 2 output variables, the data matrix data4 containing all information about the points making up the computed curve and the description variable desc4, whose contents can be shown by executing sprintf(desc4) and is also printed by the PSPMequi function to the Matlab console on exit. The PSPMequi function does produce two additional output variables containing information about bifurcations, but since no detection of such special points is carried out during computations of transcritical bifurcation boundaries, these last two output variables are empty. As shown in the Matlab 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 PNAS2002equi_demo.m script subsequently uses the first and and the sixth column of data4 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 bdata2, which was detected during the computation of the consumer-resource equilibrium without any predators (see Matlab command box 8.C). 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 bdata2 represents the birth rate of the structured consumer population in this equilibrium (see the listing of bdata2 in Matlab command box 8.C). 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 [bdata2([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 box 7.10). 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 cell array 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 Matlab command box 8.F therefore equals the cell array `{'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 8.F: Computing the predator invasion boundary

```
   >> [data5, desc5] = PSPMequi('PNAS2002', 'BPE', [bdata2([1 2 5]) 0.01], -0.1, [1 0 4.0E-4 11 0 0.1], [],...
2                          {'envBP', '1', 'envZE', '2'});

4  Executable PNAS2002equi.mexmaci64 is up-to-date

6     2.53602314E-04  8.85690312E-06  2.33333540E-06  1.00000000E-02
      2.44422274E-04  8.88032498E-06  2.35250524E-06  1.02628274E-02
8     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
12    4.02376220E-04  2.75405272E-04  1.08413352E-05  8.10619333E-02

14
   #
16 # Executing : PSPMequi('PNAS2002', 'BPE', [0.000253602 8.8569e-06 2.33334e-06 0.01], -0.1, [1 0 0.0004 11 0 0.1], [],
                      {'envBP', '1', 'envZE', '2'})
18 #
   # Parameter values  :
20 #
   # Rho      : 0.1          Rmax     : 0.000253602  Lb       : 7
22 # Lv       : 27           Lj       : 110          Lm       : 300
   # Beta     : 9E-06        Imax     : 0.0001       Rh       : 1.5E-05
24 # Gamma    : 0.006        Rm       : 0.003        Mub      : 0.01
   # A        : 5000         Th       : 0.1          Epsilon  : 0.5
26 # Delta    : 0.01
   #
28 # Index of bifurcation parameter #1                      : 1
   # Index of bifurcation parameter #2                      : 11
30 # Index of environment variable with transcritical bifurcation : 1
   #
32 #   1:Rmax     2:E[ 0]    3:E[ 1]     4:E[ 2]    5:b[ 0]     6:Mub 7:I[ 0][ 0] 8:I[ 0][ 1] 9:I[ 0][ 2]  .. 13:RHS norm
   #
34 >> data5

36 data5 =

38    0.0002536  8.8569e-06         0          0 2.3333e-06      0.01  2.4475e-05  4.008e-06  0.00041897  ..  5.083e-08
      0.00024442 8.8803e-06         0          0 2.3525e-06  0.010263  2.3554e-05  4.008e-06  0.00039918  ..  1.7842e-08
40    0.00023541 8.9071e-06         0          0 2.3732e-06  0.010542  2.2651e-05  4.008e-06  0.00037982  ..  2.3654e-08
   <...output lines suppressed in this box...>
42    0.00038848 0.00026176         0          0 1.0806e-05  0.080784  1.2672e-05  4.008e-06  5.4416e-05  ..  1.4571e-09
      0.00039542 0.00026857         0          0 1.0824e-05  0.080926  1.2685e-05  4.008e-06  5.4388e-05  ..  1.5249e-09
44    0.00040238 0.00027541         0          0 1.0841e-05  0.081062  1.2697e-05  4.008e-06  5.4362e-05  ..  1.5871e-09
```

(Once again, consult your Matlab 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 Matlab 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 (Matlab command box 8.E). The data about the points making up the computed curve are stored in the variable `data5`, whereas the description of the computations is stored in `desc5`. The last two, empty output quantities returned by the `PSPMequi` function are ignored as before. The `PNAS2002equi_demo.m` script again uses the first and and the sixth column

of `data5` 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 `bdata3`, which was detected during the computation of the predator-consumer-resource equilibrium (see Matlab command box 8.D). 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 `[bdata3([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 an empty cell array `{}` is passed as the last argument to the `PSPMequi` function, because all variables influence the location of the limit point and hence none of the quantities are characterised by a zero equilibrium state.

Command box 8.G: Computing the predator persistence boundary

```
   >> [data6, desc6] = PSPMequi('PNAS2002', 'LP', [bdata3([1:5]) 0.01], 0.1, [1 0 4.0E-4 11 0 0.1], [], {});
2
   Executable PNAS2002equi.mexmaci64 is up-to-date
4
   8.84988526E-05  1.31478259E-05  3.80697042E-05  4.00801603E-06  5.17387220E-05  1.00133536E-02
6  8.88061600E-05  1.32276426E-05  3.77736397E-05  4.00801603E-06  5.08693879E-05  1.02496482E-02
   8.91170085E-05  1.33078344E-05  3.74734505E-05  4.00801603E-06  5.00036622E-05  1.04892017E-02
8  <...output lines suppressed in this box...>
   1.12736550E-04  1.70612462E-05  9.68666652E-08  4.00801603E-06  4.97533817E-06  3.44553531E-02
10 1.12762559E-04  1.70588288E-05  2.96439331E-09  4.00801603E-06  4.94135869E-06  3.45011382E-02
   1.12788464E-04  1.70563741E-05 -9.10003572E-08  4.00801603E-06  4.90755125E-06  3.45468972E-02
12

14 #
   # Executing : PSPMequi('PNAS2002', 'LP', [8.84989E-05 1.32327E-05 3.8454E-05 4.00802E-06 5.25658E-05 0.01], 0.1,
16 #                 [1 0 0.0004 11 0 0.1], [], {})
   #
18 # Parameter values  :
   #
20 # Rho      : 0.1         Rmax     : 8.84989E-05  Lb       : 7
   # Lv       : 27          Lj       : 110          Lm       : 300
22 # Beta     : 9E-06        Imax     : 0.0001       Rh       : 1.5E-05
   # Gamma    : 0.006        Rm       : 0.003        Mub      : 0.01
24 # A        : 5000         Th       : 0.1          Epsilon  : 0.5
   # Delta    : 0.01
26 #
   # Index of bifurcation parameter #1                       : 1
28 # Index of bifurcation parameter #2                       : 11
   #
30 #    1:Rmax    2:E[ 0]     3:E[ 1]     4:E[ 2]     5:b[ 0]     6:Mub 7:I[ 0][ 0] 8:I[ 0][ 1] 9:I[ 0][ 2] ..  13:RHS norm
   #
32 >> data6

34 data6 =

36 8.8499e-05  1.3148e-05   3.807e-05  4.008e-06  5.1739e-05   0.010013  7.5351e-06  4.008e-06  6.4614e-05  ..   1.6208e-10
   8.8806e-05  1.3228e-05  3.7774e-05  4.008e-06  5.0869e-05    0.01025  7.5579e-06  4.008e-06  6.5012e-05  ..   4.3746e-12
38 8.9117e-05  1.3308e-05  3.7473e-05  4.008e-06  5.0004e-05   0.010489  7.5809e-06  4.008e-06  6.5406e-05  ..    1.499e-10
   <...output lines suppressed in this box...>
40 0.00011274  1.7061e-05  9.6867e-08  4.008e-06  4.9753e-06   0.034455  9.5675e-06  4.008e-06   8.158e-05  ..   4.4366e-12
   0.00011276  1.7059e-05  2.9644e-09  4.008e-06  4.9414e-06   0.034501  9.5704e-06  4.008e-06  8.1585e-05  ..    9.821e-12
42 0.00011279  1.7056e-05     -9.1e-08  4.008e-06  4.9076e-06   0.034547  9.5732e-06  4.008e-06   8.159e-05  ..    1.219e-12
```

(Once again, consult your Matlab 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 (Matlab command box 8.E and 8.F). The data about the points making up the computed curve are stored in the variable `data6`, whereas the description of the computations is stored in `desc6`. The last two, empty output quantities returned by the `PSPMequi` function are ignored as before. The `PNAS2002equi_demo.m` script again uses the first and and the sixth column of `data6` 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.

8.5   Output files generated by the `PSPMequi` script

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 case of two-parameter continuations (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 `.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`, `.mat` 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 Matlab command box 8.B generates the output files `PNAS2002-EQ-0000.bif`, `PNAS2002-EQ-0000.err`, `PNAS2002-EQ-0000.out` and `PNAS2002-EQ-0000.mat`, the following call as shown in Matlab command box 8.C generates the output files `PNAS2002-EQ-0001.bif`, `PNAS2002-EQ-0001.err`, `PNAS2002-EQ-0001.out` and `PNAS2002-EQ-0001.mat`, while the last computation of an equilibrium curve, as shown in Matlab command box 8.D, generates the output files `PNAS2002-EQ-0002.bif`, `PNAS2002-EQ-0002.err`, `PNAS2002-EQ-0002.out` and `PNAS2002-EQ-0002.mat`. The computations of the consumer invasion, predator invasion and predator persistence boundary (see Matlab command box 8.E, 8.F and 8.G) each generate only 3 output files, called `PNAS2002-<Type>-0000.err`, `PNAS2002-<Type>-0000.out` and `PNAS2002-<Type>-0000.mat` 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 is contained in the first two output variables `curvepoints` and `curvedesc` returned by the `PSPMequi` function (see Matlab command box 8.I). 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 `sprintf(curvedesc)`. Following this descriptive header the file contains columns with computational results that are also contained in the variable `curvepoints` (see, for example, Matlab command box 8.C). In fact, the first two output variables `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 a single string variable `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 output variables `bifpoints` and `biftypes` returned by the `PSPMequi` function (see Matlab command box 8.I). Each row in the file `<Modelname>-<Type>-<NNNN>.bif` pertains to a single detected bifurcation point. A row starts with the numerical data that characterises 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 output variable `bifpoints`, which hence has as many columns as there are in the output variable `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 the output cell array `biftypes`, which hence has as many cells as there are bifurcation points.

The file called `<Modelname>-<Type>-<NNNN>.mat` 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. For example, the file `PNAS2002-EQ-0002.mat` is generated by the invocation of the `PSPMequi` function for the PNAS model shown in Matlab command box 8.D. Highlighting (selecting) this file in Matlab's "Current folder" window displays its contents in Matlab's "File details" window. It consists of a series of population states, one for each of the points computed along the curve, as shown in the following box (notice that some intermediate output is suppressed as was also the case in previous Matlab command boxes):

```
    State_2_536023E_04 <1x1 struct>
2   State_2_525758E_04 <1x1 struct>
    State_2_515530E_04 <1x1 struct>
4   <...output lines suppressed in this box...>
    State_3_898128E_04 <1x1 struct>
6   State_3_970218E_04 <1x1 struct>
    State_4_042258E_04 <1x1 struct>
```

The population state called `State_4_042258E_04` pertains to the parameter value $R_{max} = 4.042258 \cdot 10^{-4}$ as its name suggests. Loading this state into the Matlab workspace reveals its contents to be various arrays of numbers, as shown in the following box:

Command box 8.H: Contents of population state generated by `PSPMequi`

```
    >> load('PNAS2002-EQ-0002.mat', 'State_4_042258E_04')
2   >> State_4_042258E_04

4   State_4_042258E_04 =
```

```
 6                    BifPars: 0.00040423
                   Parameters: [0.1 0.00040423 7 27 110 300 9e-06 0.0001 1.5e-05 0.006 0.003 0.01 5000 0.1 0.5 0.01]
 8                Environment: [0.00025335 0.0001336 4.008e-06]
          Pop00_BirthStates: [0 7]
10                       Pop00: [100x3 double]
```

The first element (called `BifPars`) of the structure containing the population state
`State_4_042258E_04` is 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 state has been computed, while the fourth member
of the structure `State_4_042258E_04` contains the equilibrium values of all environment
variables. The two subsequent arrays 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 distri-
bution in the equilibrium point with the first column `State_4_042258E_04.Pop00(:,1)`
representing the density profile of the equilibrium population and the subsequent columns
`State_4_042258E_04.Pop00(:,2)` and `State_4_042258E_04.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 below:

```
   >> State_4_042258E_04.Pop00
 2
   ans =
 4
       0.00043495        1.475        9.4233
 6     7.1561e-07        18.427       35.856
       6.3189e-07        30.804       53.561
 8 <...output lines suppressed in this box...>
       4.9656e-12        1206.2       283.03
10     4.3877e-12        1218.6       283.05
       3.8771e-12        1230.9       283.06
```

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 Matlab com-
mand box above also illustrates the contents of the equilibrium population distribution by
listing the first and last 3 rows of the two-dimensional array `State_4_042258E_04.Pop00`.
The dimension of the array `State_4_042258E_04.Pop00` indicates that the population
is represented by 100 cohorts of individuals (see chapter 17 for the option to change
this number). The number of individuals in cohort i is given by the array element
`State_4_042258E_04.Pop00(i,1)`, while the average value of the individual state vari-
able with index 0 and 1 (average age and average length in the PNAS model) are given by
`State_4_042258E_04.Pop00(i,2)` and `State_4_042258E_04.Pop00(i,3)`, respectively.

Command box 8.I: `PSPMequi help page`

```
 1    >> help PSPMequi
 2      PSPMequi: Computes a bifurcation curve for a structured population model
 3
 4      Syntax:
 5
 6        [curvepoints, curvedesc, bifpoints, biftypes] = ...
 7            PSPMequi(modelname, biftype, startpoint, stepsize, parbnds, parameters, options,...
 8                  'clean', 'force', 'debug')
 9
10      Arguments:
11
12        modelname:  (string, required)
13                    Basename of the file with model specification. The file
14                    should have extension '.h'. For example, the model 'PNAS2002'
15                    is specified in the file 'PNAS2002.h'
16
17        biftype:    (string, required)
18                    Type of bifurcation to compute: BP, BPE, EQ, LP, ESS or PIP
19
20        startpoint: (row vector, required)
21                    The initial point from which to start the continuation of
22                    the curve
23
24        stepsize:   (double value, required)
25                    Value of the step size in the first bifurcation parameter
26
27        parbnds:    (row vector of length n*3, required)
28                    Vector of length n*3 with n=1 for EQ continuation, n=2 for
29                    BP, BPE, LP and PIP continuation and n >=2 for ESS continuation.
30                    Each triples specifies:
31
32                    parbnds(1): the index of the first bifurcation parameter
33                    parbnds(2): lower threshold, below which value of the
34                                first bifurcation parameter the computation stops
35                    parbnds(3): upper threshold, above which value of the
36                                first bifurcation parameter the computation stops
37
38                    In case of two-parameter bifurcations:
39
40                    parbnds(4): the index of the second bifurcation parameter
41                    parbnds(5): lower threshold, below which value of the
42                                second bifurcation parameter the computation stops
43                    parbnds(6): upper threshold, above which value of the
44                                second bifurcation parameter the computation stops
45
46        parameters: (row vector, required, but can be the empty vector [])
47                    Vector of length PARAMETER_NR (set in the model program
48                    file), specifying the values for the model parameters to
49                    use in the computation. Vectors of other lengths, including
50                    an empty vector will be ignored.
51
52        options:    (cell array, required, but can be the empty cell array {})
53                    Cell array with pairs of an option name and a value (for
54                    example {'popBP', '1'}) or single options (i.e. 'test').
55                    Possible option names and their values are:
56
57                    'envBP',  '<index>': Index of environment variable, of which
58                                         to continue the transcritical bifurcation
59                    'popBP',  '<index>': Index of structured population, of which
60                                         to continue the transcritical bifurcation
61                    'popEVO', '<index>': Index of structured population, for
62                                         which to compute the selection gradient or
63                                         perform ESS or PIP continuation
64                    'parEVO', '<index>': Index of parameter, for which to compute
65                                         the selection gradient
66                    'envZE',  '<index>': Index of environment variable in
67                                         trivial equilibrium (can be used
68                                         multiple times)
69                    'popZE',  '<index>': Index of structured population in
70                                         trivial equilibrium (can be used
71                                         multiple times)
72                    'isort',  '<index>': Index of i-state variable to use as
73                                         ruling variable for sorting the
74                                         structured populations
75                    'noBP'             : Do not check for branching points while
76                                         computing equilibrium curves
77                    'noLP'             : Do not check for limit points while
78                                         computing equilibrium curves
79                    'single'           : Only compute the first point of the
80                                         solution curve, do not continue the curve
81                    'test'             : Perform only a single integration over
82                                         the life history, reporting dynamics
83                                         of survival, R0, i-state and
84                                         interaction variables
85
86        'clean':    (string. optional argument)
87                    Remove all the result files of the model before the
88                    computation
89
90        'force':    (string, optional argument)
91                    Force a rebuilding of the model before the computation
92
93        'debug':    (string, optional argument)
```

```
 94                    Compile the model in verbose mode and with debugging flag set

 96      Output:

 98        curvepoints: Matrix with output for all computed points along the curve

100        curvedesc:   Column vector with strings, summarizing the numerical details
                        of the computed curve (i.e., initial point, parameter values,
102                     numerical settings used)
          bifpoints:   Matrix with the located bifurcation points along the curve
104
          biftypes:    Column vector of strings, containing a description of the
106                     type of bifurcation for each of the located bifurcation points
```

Evolutionary analysis of non-linear PSPMs

Theoretical and computational background

The analysis 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

$$\left.\frac{\partial s_x(y)}{\partial y}\right|_{y=x} > 0 \tag{9.1}$$

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

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

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:

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

x in this condition 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 chapter 8. 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 8.2). As discussed in section 8.2 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 characterised 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.

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 $n(t, s)$:

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

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

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

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:

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

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

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

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

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` that you can find in the `Tests` directory. The implementation of the model follows the guidelines as presented for the PNAS model in chapter 7 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.

Model analysis in Matlab

The analysis using Matlab can be performed by executing the Matlab script `Indet_growth_demo.m`, which is to be found together with the model-specific file `Indet_growth.h` in the `Tests` directory. Below 3 commands will be discussed that are executed by the `Indet_growth_demo.m` script and that illustrate the possibilities to use the software for evolutionary fixed point analysis. The `Indet_growth_demo.m` script 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 `[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 Matlab command box below.

Furthermore, the computations are carried out with the default parameter values, such that the 6th argument of the function `PSPMequi` is an empty vector (`[]`). The last argument of the function `PSPMequi` is the option cell array, 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 11.A: Detection of an evolutionary fixed point

```
   >> [data1, desc1, bdata1, btype1] = PSPMequi('Indet_growth', 'EQ', [1.0 0.22 0.0], -0.1, [6 0.5 2.0], [], {'popEVO', '0'});

   Building executable Indet_growthequi.mexmaci64 ...

    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', [1 0.22 0], -0.1, [6 0.5 2], [], {'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
   #
   >> data1

   data1 =

            1       0.22     0.035537      0.178      0.5323     0.27679      1      -41.446    9.5791e-09
      0.99804    0.21965     0.034538    0.17803     0.52939     0.27986      1      -38.946    3.2943e-08
      0.99601    0.21931     0.033539    0.17807     0.52629     0.28312      1      -36.481    3.0528e-08
   <...output lines suppressed in this box...>
      0.94457    0.21562     0.018554    0.17844     0.43178      0.3793      1      -2.0917    1.6189e-08
      0.93826    0.21559     0.017589    0.17844     0.41944     0.39166      1  -9.0639e-11    7.9831e-09
      0.93803    0.21559     0.017556    0.17844     0.41899      0.3921      1     0.070595    8.0441e-09
   <...output lines suppressed in this box...>
      0.51867    0.24053     0.007045    0.17595     0.10149     0.69826      1       6.4213    1.466e-09
      0.50928    0.24121    0.0070112    0.17588    0.099431     0.70002      1       6.3229    2.2549e-09
      0.49986    0.24189    0.0069786    0.17581    0.097426     0.70171      1        6.227    1.9907e-09

   >> bdata1

   bdata1 =

      0.93826    0.21559     0.017589    0.17844     0.41944     0.39166      1  -9.0639e-11    7.9831e-09

   >> btype1

   btype1 =

       '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 cell array `btype1` 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 `data1` produced by the call to `PSPMequi`, has an additional column of output compared to the columns of output discussed in chapter 8, 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 Matlab script `Indet_growth_demo.m` 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 start-
ing 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 Matlab 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 popula-
tion birth rate(s). Since the default parameter value for p is 1.0, the initial point of the
`'ESS'` computation below is specified as `[1.0 bdata1([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
`[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 11.A the
option cell array that is supplied to the `PSPMequi` function as 7th argument equals
`{'popEVO', '0'}` to indicate that the evolutionary computations should focus on the
R_0 value of the structured population with index 0.

Command box 11.B: Continuation of an evolutionary fixed point

```
   >> [data2, desc2] = PSPMequi('Indet_growth', 'ESS', [1.0 bdata1([2 3 1])], 0.1, [9 0.5 2.0 6 0.5 2.0], [], {'popEVO', '0'});

   Executable Indet_growthequi.mexmaci64 is up-to-date

     1.00000000E+00  2.15592978E-01  1.75885671E-02  9.38263361E-01
     1.04628202E+00  2.15928682E-01  1.83620718E-02  9.81458017E-01
     1.09100600E+00  2.16276945E-01  1.91532469E-02  1.02305121E+00
   <...output lines suppressed in this box...>
     1.97281065E+00  2.31687764E-01  4.44253948E-02  1.79193220E+00
     1.99862418E+00  2.32535490E-01  4.53663299E-02  1.81213728E+00
     2.02444044E+00  2.33417049E-01  4.63073461E-02  1.83215556E+00


   #
   # Executing : PSPMequi('Indet_growth', 'ESS', [1 0.215593 0.0175886 0.938263], 0.1, [9 0.5 2 6 0.5 2], [], {'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  5:I[ 0][ 0] .. 8:R0[ 0]  9:R0_x[ 9]  10:R0_xx[ 0]  11:R0_yy[ 0]  12:RHS norm
   #
   >> data2

   data2 =

         1    0.21559   0.017589     0.93826      0.17844 ..      1     -1.5739    0.00013642   -0.00013624   1.8636e-07
      1.0463    0.21593   0.018362     0.98146      0.17841 ..      1     -1.6704    0.00014653   -0.00014642   4.2147e-10
```

```
36        1.091   0.21628  0.019153     1.0231      0.17837  ..      1   -1.7681    0.00015623   -0.00015609   6.4441e-10
         <...output lines suppressed in this box...>
38        1.9728  0.23169  0.044425     1.7919      0.17683  ..      1   -4.6071    0.00026926   -0.00026795   3.2118e-10
          1.9986  0.23254  0.045366     1.8121      0.17675  ..      1   -4.7119    0.00026945   -0.00026812   3.4521e-10
40        2.0244  0.23342  0.046307     1.8322      0.17666  ..      1   -4.8174    0.00026949   -0.00026813   4.1015e-10
```

(Notice that the intermediate columns specifying the value of some of the interaction variables in equilibrium have been suppressed as output in the Matlab 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 data2 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 chapter 9 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 Matlab script Indet_growth_demo.m for this computation consists of the curve of the evolutionary fixed point value of q (column 4 in the data matrix data2) as a function of the bifurcation parameter p (column 1 in the data matrix data2).

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 *et al.* (2015).

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 bdata1([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 [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 11.C: Construction of a pairwise invasibility plot

```
 2    >> data3 = PSPMequi('Indet_growth', 'PIP', bdata1([1 2 3 1]),  0.1, [6 0.5 2.0 6 0.5 2.0], [], {'popEVO', '0'}),
         data4 = PSPMequi('Indet_growth', 'PIP', bdata1([1 2 3 1]), -0.1, [6 0.5 2.0 6 0.5 2.0], [], {'popEVO', '0'})

 4    Executable Indet_growthequi.mexmaci64 is up-to-date

 6       9.38263361E-01  2.15592978E-01  1.75885671E-02  9.38263361E-01
         9.44757579E-01  2.15623452E-01  1.85841380E-02  9.31442089E-01
 8       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
12       1.05878535E+00  2.41902724E-01  8.31367021E-02  4.99682764E-01

14
      #
16    # Executing : PSPMequi('Indet_growth', 'PIP', [0.938263 0.215593 0.0175886 0.938263], 0.1, [6 0.5 2 6 0.5 2], [],
                          {'popEVO', '0'})
18    #
      # Parameter values  :
20    #
      # Delta    : 0.1          Rmax     : 2          Sb       : 0.05
22    # Sj       : 1            Sm       : 2          Imax     : 1
      # q        : 0.938263     Sigma    : 0.5        T        : 0.1
24    # p        : 1            Mu       : 0.01
      #
26    # Index of bifurcation parameter #1                         : 6
      # Index of structured population for evolutionary analysis  : 0
28    #
      #         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    #

32    data3 =

34           0.93826      0.21559      0.017589      0.93826      0.17844      0.41944      0.39166         1        1    1.462e-07
             0.94476      0.21562      0.018584      0.93144      0.17844      0.43215      0.37893         1        1    6.9554e-10
36           0.95052      0.21571       0.01958      0.92477      0.17843      0.44349      0.36755         1        1    4.0596e-09
      <...output lines suppressed in this box...>
38           1.0572      0.24098      0.081604       0.5125       0.1759      0.59762      0.20193         1        1    9.5157e-12
             1.058      0.24144      0.082372      0.50611      0.17586      0.59846      0.20089         1        1    1.0016e-11
40           1.0588       0.2419      0.083137      0.49968      0.17581      0.59929      0.19985         1        1    1.0534e-11

42    Executable Indet_growthequi.mexmaci64 is up-to-date

44       9.38263361E-01  2.15592978E-01  1.75885671E-02  9.38263361E-01
         9.30859689E-01  2.15628723E-01  1.65937656E-02  9.45282279E-01
46       9.22302855E-01  2.15749165E-01  1.56001074E-02  9.52520009E-01
      <...output lines suppressed in this box...>
48       5.13691984E-01  2.40890443E-01  7.02691650E-03  1.05701563E+00
         5.04282772E-01  2.41569757E-01  6.99375827E-03  1.05820719E+00
50       4.94835713E-01  2.42254183E-01  6.96170534E-03  1.05939157E+00

52
      #
54    # Executing : PSPMequi('Indet_growth', 'PIP', [0.938263 0.215593 0.0175886 0.938263], -0.1, [6 0.5 2 6 0.5 2], [],
                          {'popEVO', '0'})
56    #
      # Parameter values  :
58    #
      # Delta    : 0.1          Rmax     : 2          Sb       : 0.05
60    # Sj       : 1            Sm       : 2          Imax     : 1
      # q        : 0.938263     Sigma    : 0.5        T        : 0.1
62    # p        : 1            Mu       : 0.01
      #
64    # Index of bifurcation parameter #1                         : 6
      # Index of structured population for evolutionary analysis  : 0
66    #
      #         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
68    #

70    data4 =

72           0.93826      0.21559      0.017589      0.93826      0.17844      0.41944      0.39166         1        1    1.462e-07
             0.93086      0.21563      0.016594      0.94528      0.17844      0.40513      0.40595         1        1    2.609e-10
74           0.9223      0.21575       0.0156      0.95252      0.17843      0.38898      0.42204         1        1    9.0182e-09
      <...output lines suppressed in this box...>
76           0.51369      0.24089      0.0070269       1.057      0.17591      0.10039       0.6992         1        1    1.0273e-07
             0.50428      0.24157      0.0069938      1.0582      0.17584      0.098359      0.70093         1        1    1.0158e-07
78           0.49484      0.24225      0.0069617      1.0594      0.17577      0.096386      0.70259         1        1    1.1849e-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 q and q', , the resident and the mutant value of the evolutionary parameter, 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 Matlab script Indet_growth_demo.m uses the first and fourth output columns, corresponding to the two bifurcation parameters, from the

data matrices `data3` and `data4` that result from the first and second call to the function `PSPMequi` in the command box above to construct the pairwise invasibility plot (PIP).

The Matlab script `Indet_growth_demo.m` furthermore verifies the construction of the PIP by computing the boundary that are contained in the data matrices `data3` and `data4` via an alternative route. For this purpose it uses the model-specific file `Indet_growth_resmut.h`, which implements the same model as specified in `Indet_growth.h`, but explicitly accounts for the resident and the mutant type as two independent structured populations that compete for the same resource. Using this implementation with two populations, the boundary in the pairwise invasibility plot that separates parameter regions with a positive and negative population growth rate of the mutant type, corresponds to the transcritical bifurcation curve (or invasion boundary) of the mutant population into the resident population. The Matlab script `Indet_growth_demo.m` computes this transcritical bifurcation curve by taking the last computed equilibrium point from the data matrix `data4` and using it as initial point for the following command:

```
[data5, desc5] = PSPMequi('Indet_growth_resmut', 'BP', data4(end,1:4), 0.1, [6 0.5 2.0 12 0.5 2.0], [], {'popBP', '1'});
```

The plot that is subsequently generated by the `Indet_growth_demo.m` script confirms the correctness of the curve.

Simulating evolutionary dynamics

12.1 Theoretical background

In the context of adaptive dynamics the change over evolutionary time in a set of life history traits, characterising the individuals of a population, can be described by the so-called *canonical equation* (Dieckmann & 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,

$$\frac{d\mathbf{x}}{dt} \;=\; n_e(\mathbf{x})\, \theta\, \boldsymbol{\Sigma}\, \left.\frac{\partial s_{\mathbf{x}}(\mathbf{y})}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{x}} \tag{12.1}$$

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 summarising 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 (9.1)). As discussed in section 9 the selection gradient is sign–equivalent with the following derivative of R_0:

$$\left.\frac{\partial R_0(\mathbf{y}, \tilde{E}(\mathbf{x}))}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{x}} \tag{12.2}$$

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 chapters 9-11. 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:

$$\frac{d\mathbf{x}}{dt} \;\propto\; \tilde{b}(\mathbf{x})\, \boldsymbol{\Sigma}\, \left.\frac{\partial R_0(\mathbf{y}, \tilde{E}(\mathbf{x}))}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{x}} \tag{12.3}$$

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 analysing 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` scripts a script called `PSPMevodyn` to simulate the change in an arbitrary number of life history parameters over evolutionary time. As a starting point the script 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 (12.3) that is proportional to the evolutionary rate of change in the life history parameters as determined by the canonical equation. Unless explicitly specified, the script 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.

12.2   Simulating evolutionary dynamics in Matlab

The use of the `PSPMevodyn` script will be illustrated with the same model as described in chapter 10 and analysed in chapter 11. In fact, the Matlab script `Indet_growth_demo.m`, that is to be found together with the model-specific file `Indet_growth.h` in the `Tests` directory and that was already discussed in the previous chapter, performs at the end 2 computations of trait dynamics over evolutionary time. The `Indet_growth_demo.m` script furthermore performs some plotting of the output data generated by these computations.

The general call to the `PSPMevodyn` script is shown in the command box below.

Command box 12.A: General syntax of a `PSPMevodyn` call

```
2    >> [curvepoints, curvedesc] = ...
                     PSPMevodyn(modelname, startpoint, curvepars, evopars, covars, parameters, options,...
                               'clean', 'force', 'debug')
```

The obligatory arguments to the `PSPMevodyn` script are the following:

1. The first argument to the Matlab-script `PSPMevodyn` is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension `.h` as part of the file name, the Matlab-script will strip the `.h` extension away if it is included. The Matlab-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.

2. The second 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 Matlab (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:

   [*<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 argument to the `PSPMevodyn` function is a Matlab 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 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 Matlab (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:

[*<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:

[*<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 argument of the `PSPMevodyn` function specifies the variance–covariance matrix Σ (see equation (12.1)). 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 empty vector is specified the matrix Σ is taken equal to the identity matrix.

6. The sixth obligatory argument of the `PSPMevodyn` function is a Matlab (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (`PARAMETER_NR`). 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 `PARAMETER_NR`, for example when an empty vector is passed as the sixth argument, it will simply be ignored.

7. The seventh and last obligatory argument of the `PSPMevodyn` function is a `Matlab` cell array containing possible options that modify the behaviour 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 `Matlab` where array indices start at 1).

   - Option pair `{'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.

   - Option pair `{'envZE', 'i'}`: This option pair can be specified several times as part of the option cell array. 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 box 7.12) 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 7.3.2).

   - Option pair `{'popZE', 'i'}`: This option pair can be specified several times as part of the option cell array. 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`).

   - Option pair `{'isort', 'i'}`: This option modifies the output of the population states that are stored in an output file with a name of the form `<Modelname>-EVODYN-<NNNN>.mat`. By default the computational module reports the information about the 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 `{'isort', '0'}` as option cell array 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 `{'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 chapter 17).

- Option `{'test'}`: The last possible option that can be passed to the `PSPMevodyn` function as part of the option cell array 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'`. Unlike the previous arguments, which all modify the computations to be performed, these options modify the behaviour of the `PSPMevodyn` function itself, in particular the compilation of the model specific file into a `mex` module that can be executed from `Matlab`. 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'`: This optional argument instructs the `PSPMevodyn` function to delete all result files that have been generated during previous calculations with the model, i.e. as a result of previous calls to `PSPMdemo`, `PSPMequi` or `PSPMevodyn`. These result files have names of the form `<Modelname>-<Type>-<NNNN>.bif`, `<Modelname>-<Type>-<NNNN>.err`, `<Modelname>-<Type>-<NNNN>.mat` 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 contains a function `PSPMclean`, taking no arguments, to delete all `.bif`, `.err`, `.mat` and `.out` files and/or all executable files that are present in the current working directory.

- Option `'force'`: Using the option `'force'` instructs the `PSPMevodyn` file to force re-compilation of the model specific file into a `mex` module that can be executed by `Matlab`. 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 `mex` file. However, if for some unclear reason this automatic recompilation fails, the `'force'` option can be used to initiate re-compilation.

- Option `'debug'`: This option instructs the `PSPMevodyn` function to turn on debugging flags while compiling the model specific file into a `mex` 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 `Matlab` 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 2 output variables on execution (see the help page on PSPMevodyn in Matlab command box 12.D). The first output variable curvepoints contains the numerical information of the points along the computed curve. This variable 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.

When the PSPMevodyn function finishes, it prints textual information about the computation that has been carried out. This textual information is also contained in the second output variable curvedesc of the function (see box 12.D). Among others, this text contains a header line indicating which column of the output contains which particular value (see Matlab command box 12.B below for an example). The PSPMevodyn function prints its report on the calculations by execution of the statement sprintf(curvedesc).

12.3   An example session using the PSPMevodyn script

The Matlab script Indet_growth_demo.m 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 chapter 10). The particular call to the PSPMevodyn function is shown in the following Matlab command box:

Command box 12.B: Simulating evolutionary dynamics in a single parameter

```
     >> [data1, desc1] = PSPMevodyn('Indet_growth', [0.22 0.03554 1.0], [0.05 10], [6 0.5 1.5], [], [], {'popEVO', '0'});
 2
     Building executable Indet_growthevodyn.mexmaci64 ...
 4
      0.005000     2.20000000E-01  3.55373787E-02  1.00000000E+00
 6    0.010000     2.18793851E-01  3.19778211E-02  9.92635669E-01
      0.015000     2.18089189E-01  2.97666950E-02  9.87408855E-01
 8   <...output lines suppressed in this box...>
      2.581562     2.15592977E-01  1.75885665E-02  9.38263363E-01
10    2.631562     2.15592977E-01  1.75885665E-02  9.38263362E-01
      2.681562     2.15592977E-01  1.75885664E-02  9.38263362E-01
12

14   #
     # Executing : PSPMevodyn('Indet_growth', [0.22 0.03554 1], [0.05 10], [6 0.5 1.5], [], [], {'popEVO', '0'})
16   #
     # Parameter values  :
18   #
     # Delta    : 0.1          Rmax    : 2          Sb      : 0.05
20   # Sj       : 1            Sm      : 2          Imax    : 1
     # q        : 1            Sigma   : 0.5        T       : 0.1
22   # p        : 1            Mu      : 0.01
     #
24   # Index of structured population for evolutionary dynamics   : 0
     # Index of evolution parameter #0                            : 6
26   #
     # 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
28   #
     >> data1
```

```
 30
      data1 =

 32
             0.005         0.22      0.035537            1         0.178        0.5323       0.27679            1   9.5869e-09
 34          0.01       0.21879      0.031978      0.99264       0.17812       0.52101       0.28863            1   8.7718e-09
            0.015       0.21809      0.029767      0.98741       0.17819       0.51246        0.2975            1   3.5224e-09
 36   <...output lines suppressed in this box...>
            2.5816       0.21559      0.017589      0.93826       0.17844       0.41944       0.39166            1   2.3086e-07
 38          2.6316       0.21559      0.017589      0.93826       0.17844       0.41944       0.39166            1   2.3086e-07
            2.6816       0.21559      0.017589      0.93826       0.17844       0.41944       0.39166            1   2.3086e-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 stabilised 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 Σ (refer to equation (12.1)) 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 cell array with options, 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 Matlab command box above the values of the computed points along the evolutionary trajectory are stored in the variable data1, the contents of which are inspected after the PSPMevodyn function finishes and it has printed out the textual information about the computation. The demonstration script Indet_growth_demo.m uses the data contained in the first and the fourth column of data1 to plot the time course of evolutionary change in the parameter q.

In the following Matlab 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 chapter 10), 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 Matlab command box 12.B.

Command box 12.C: Simulating evolutionary dynamics in two parameters

```
    >> [data2, desc2] = PSPMevodyn('Indet_growth',[0.22 0.03554 1.0 1.0],[0.05 100],[6 0.5 1.5 9 0.5 1.5],[],[],{'popEVO','0'});
 2
    Executable Indet_growthevodyn.mexmaci64 is up-to-date
 4
    0.005000    2.20000000E-01  3.55373787E-02  1.00000000E+00   1.00000000E+00
 6  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
 8  <...output lines suppressed in this box...>
    59.960312   2.13281602E-01  1.20360784E-02  4.99880243E-01   5.34824255E-01
10  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
12

14  #
    # Executing : PSPMevodyn('Indet_growth', [0.22 0.03554 1 1], [0.05 100], [6 0.5 1.5 9 0.5 1.5], [], [], {'popEVO', '0'})
16  #
    # Parameter values  :
18  #
    # Delta    : 0.1        Rmax     : 2          Sb       : 0.05
20  # Sj       : 1          Sm       : 2          Imax     : 1
    # q        : 1          Sigma    : 0.5        T        : 0.1
22  # p        : 1          Mu       : 0.01
    #
24  # Index of structured population for evolutionary dynamics  : 0
    # Index of evolution parameter #0                           : 6
26  # Index of evolution parameter #1                           : 9
    #
28  # 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
    #
30  >> data2

32  data2 =

34       0.005       0.22     0.035537         1        1      0.178      0.5323     0.27679        1   9.5869e-09
             0.01      0.218     0.029583   0.99264   1.0067     0.1782     0.51299     0.30033        1   3.9863e-09
36           0.015    0.21728     0.027093   0.98875   1.0101    0.17827     0.50136     0.31348        1   3.4821e-09
    <...output lines suppressed in this box...>
38          59.96     0.21328     0.012036   0.49988  0.53482    0.17867      0.3555     0.34742        1   7.8664e-07
            60.01     0.21328     0.012036   0.49988  0.53482    0.17867      0.3555     0.34742        1   7.8664e-07
40          60.06     0.21328     0.012036   0.49988  0.53482    0.17867      0.3555     0.34742        1   7.8664e-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.

12.4 Output files generated by the PSPMevodyn script

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 .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, .mat 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 output variables `curvepoints` and `curvedesc` returned by the `PSPMevodyn` function (see box 12.D). 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 `sprintf(curvedesc)`. Following this descriptive header the file contains columns with computational results that are also contained in the variable `curvepoints` (see, for example, Matlab command box 12.B). In fact, the two output variables `curvepoints` and `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 `curvedesc`, while storing the information on all other lines into the data matrix `curvepoints`.

The file called `<Modelname>-EVODYN-<NNNN>.mat` 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. For example, the file `Indet_growth-EVODYN-0001.mat` is generated by the invocation of the `PSPMevodyn` function in Matlab command box 12.C. Highlighting (selecting) this file in Matlab's "Current folder" window displays its contents in Matlab's "File details" window. It consists of a series of population states, one for each of the points computed along the evolutionary trajectory, as shown in the following box (notice that some intermediate output is suppressed as was also the case in previous Matlab command boxes):

```
    State_5_000000E_03 <1x1 struct>
2   State_1_000000E_02 <1x1 struct>
    State_1_500000E_02 <1x1 struct>
4   <...output lines suppressed in this box...>
    State_5_996031E01 <1x1 struct>
6   State_6_001031E01 <1x1 struct>
    State_6_006031E01 <1x1 struct>
```

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. Loading this state into the Matlab workspace reveals its contents to be various arrays of numbers, as shown in the following box:

```
    >> load('Indet_growth-EVODYN-0001.mat', 'State_1_500000E_02')
2   >> State_1_500000E_02

4   State_1_500000E_02 =

6              EvoTime: 0.015
               EvoPars: [0.98875 1.0101]
8            Parameters: [0.1 2 0.05 1 2 1 0.98875 0.5 0.1 1.0101 0.01]
           Environment: 0.21728
10     Pop00_BirthStates: [0 0.05]
                 Pop00: [100x3 double]
```

The first element (called `EvoTime`) of the structure contains 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 third member of the structure `State_1_500000E_02` contains the equilibrium values of all environment variables. The two subsequent arrays characterise the equilibrium 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 the equilibrium point with the first column `State_1_500000E_02.Pop00(:,1)` representing the density profile of the equilibrium population and the subsequent columns `State_1_500000E_02.Pop00(:,2)` and `State_1_500000E_02.Pop00(:,3)` representing the average values of the individual state variables with index 0 and 1 in the model (corresponding to individual age and size in the model implemented in `Indet_growth.h`), as shown below:

```
   >> State_1_500000E_02.Pop00

   ans =

        0.50709       10.004       0.0584
        0.41218       30.727      0.079118
        0.33503       51.451       0.10572
   <...output lines suppressed in this box...>
        9.4425e-10     2020.2       1.9858
        7.6751e-10     2040.9       1.9859
        6.2386e-10     2061.6       1.9861
```

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 Matlab command box above also illustrates the contents of the equilibrium population distribution by listing the first and last 3 rows of the two-dimensional array `State_1_500000E_02.Pop00`. The dimension of the array `State_1_500000E_02.Pop00` indicates that the population is represented by 100 cohorts of individuals (see chapter 17 for the option to change this number). The number of individuals in cohort i is given by the array element `State_1_500000E_02.Pop00($i$,1)`, while the average value of the individual state variable with index 0 and 1 (average age and average body size in the current model) are given by `State_1_500000E_02.Pop00($i$,2)` and `State_1_500000E_02.Pop00($i$,3)`, respectively.

Command box 12.D: PSPMevodyn help page

```
   >> help PSPMevodyn
2     PSPMevodyn: Computes the evolutionary dynamics for a structured population model using the canonical equation

4     Matlab syntax:

6        [curvepoints, curvedesc] = ...
              PSPMevodyn(modelname, startpoint, curvepars, evopars, covars, parameters, options,...
8                     'clean', 'force', 'debug')

10    Arguments:

12       modelname:  (string, required)
                     Basename of the file with model specification. The file
14                   should have extension '.h'. For example, the model 'PNAS2002'
                     is specified in the file 'PNAS2002.h'
16
         startpoint: (row vector, required)
18                   The initial point from which to start the simulation of the dynamics over
                     evolutionary time, including the initial values of the evolving parameters
20
         curvepars:  (row vector of length 2, required)
22                   Vector of length 2 specifying:

24                   curvepars(1): the maximum step size in evolutionary time during
                                   the integration of the canonical equation
26                   curvepars(2): the maximum evolutionary time at which to stop
                                   the integration of the canonical equation
28
         evopars:    (row vector of length n*3, required)
30                   Vector of length n*3 specifying:

32                   evopars(1): the index of the first evolution parameter
                     evopars(2): lower threshold, below which value of the
34                               first evolution parameter the computation stops
                     evopars(3): upper threshold, above which value of the
36                               first evolution parameter the computation stops
                     ......
38
                     evopars(n*3-2): the index of the last evolution parameter
40                   evopars(n*3-1): lower threshold, below which value of the
                                     last evolution parameter the computation stops
42                   evopars(n*3):   upper threshold, above which value of the
                                     last evolution parameter the computation stops
44
         covars:     (row vector, required, but can be the empty vector [])
46                   Vector of length n*n, where n is the number of evolving parameters.
                     The vector elements specify the values of the covariance matrix in
48                   the selection gradients. Vectors of other lengths, including an
                     empty vector will be ignored.
50
         parameters: (row vector, required, but can be the empty vector [])
52                   Vector of length PARAMETER_NR (set in the model program
                     file), specifying the values for the model parameters to
54                   use in the computation. Vectors of other lengths, including
                     an empty vector will be ignored.
56
         options:    (cell array, required, but can be the empty cell array {})
58                   Cell array with pairs of an option name and a value (for
                     example {'popBP', '1'}) or single options (i.e. 'test').
60                   Possible option names and their values are:

62                   'popEVO', '<index>': Index of structured population, for
                                          which to calculate the evolutionary dynamics
64                   'envZE',  '<index>': Index of environment variable in
                                          trivial equilibrium (can be used
66                                        multiple times)
                     'popZE',  '<index>': Index of structured population in
68                                        trivial equilibrium (can be used
                                          multiple times)
70                   'isort',  '<index>': Index of i-state variable to use as
                                          ruling variable for sorting the
72                                        structured populations
                     'test'             : Perform only a single integration over
74                                        the life history, reporting dynamics
                                          of survival, R0, i-state and
76                                        interaction variables

78       'clean':    (string. optional argument)
                     Remove all the result files of the model before the
80                   computation

82       'force':    (string, optional argument)
                     Force a rebuilding of the model before the computation
84
         'debug':    (string, optional argument)
86                   Compile the model in verbose mode and with debugging flag set

88    Output:

90       curvepoints: Matrix with output for all computed points along the curve

92       curvedesc:  Column vector with strings, summarizing the numerical details
                     of the computed curve (i.e., initial point, parameter values,
```

```
94                 numerical settings used)
```

Simulating the individual life history

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 `.mat` 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. Invoking the `PSPMind` function requires a slightly different syntax than the functions discussed in earlier chapters, because `PSPMind` only needs the values of environmental variables as input. The output of `PSPMind` function is a structure with similar contents as the structures that are normally stored in the `.mat` file (see section 5.2, 8.5 or 12.4 for a discussion of the `.mat` file). In fact, the program generates such a file with a name `<Modelname>-IND-<NNNN>.mat` which contains exactly one population state. As before, `<Modelname>` is the same as the name of the file specifying the model excluding its `.h` extension and `<NNNN>` is a 4-digit number that is unique for the current computation.

13.1 Arguments of the `PSPMind` function

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

Command box 13.A: General syntax of a `PSPMind` call

```
>> output = PSPMind(modelname, environment, parameters, options, 'clean', 'force', 'debug')
```

The obligatory and optional arguments to the `PSPMind` function are the following:

1. The first, obligatory argument to the **Matlab**-script `PSPMind` is the name of the file specifying the PSPM, passed as a string argument. It is unnecessary to include the extension `.h` as part of the file name, the **Matlab**-script will strip the `.h` extension away if it is included. The **Matlab**-command to simulate the life history with the model in `PNAS2002.h`, which is used for the illustration below, therefore takes `'PNAS2002'` as its first argument.

2. The second, obligatory argument is a Matlab (row) vector containing the values of the environmental variables, for which to compute the individual life history:

   [*<environment variable 1> <environment variable 2> ...*]

   The number of values specified in this vector should equal the value of the constant `ENVIRON_DIM` as it is defined in the model specification file (`.h` file). Notice, that this vector therefore does not contain any parameter values or birth rate values of the structured populations.

3. The third obligatory argument of the `PSPMind` function is a Matlab (row) vector of model parameter values. When used, this array should have the same length as the number of parameters in the model (`PARAMETER_NR`). When of this length the values will replace the default values of the parameters that are listed in the model specification file (see code box 7.2 for an example). If the array used for this third argument is not of the correct length `PARAMETER_NR`, for example when an empty vector is passed as the third argument, it will simply be ignored.

4. The fourth, optional argument of the `PSPMind` function is a Matlab cell array containing possible options that modify the behaviour of the computational module. The `PSPMind` function only recognises a single option `'isort'`. Hence, this fourth argument is either specified as an empty cell array `{}` or takes the form:

   `{'isort', 'i'}`

   This option modifies the output of the equilibrium state of the populations, which are stored in an output file with a name of the form `<Modelname>-IND-<NNNN>.mat` (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 `{'isort', '0'}` as option cell array 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 `{'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 box 7.2 in section 7.1.1 as well as chapter 17).

Three other, optional arguments can be passed to the `PSPMequi` function: `'clean'`, `'force'` and `'debug'`. Unlike the previous arguments, which all modify the computations to be performed, these options modify the behaviour of the `PSPMequi` function itself, in particular the compilation of the model specific file into a `mex` module that can be executed from Matlab. 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'`: This optional argument instructs the `PSPMequi` function to

delete all result files that have been generated during previous calculations with the model, i.e. as a result of previous calls to `PSPMequi`. These result files have names of the form `<Modelname>-<Type>-<NNNN>.bif`, `<Modelname>-<Type>-<NNNN>.err`, `<Modelname>-<Type>-<NNNN>.mat` and `<Modelname>-<Type>-<NNNN>.out`, 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. either `BP`, `BPE`, `EQ`, `ESS`, `LP` or `PIP`, 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 contains a function `PSPMclean`, taking no arguments, to delete all `.bif`, `.err`, `.mat` and `.out` files and/or all executable files that are present in the current working directory.

- Option `'force'`: Using the option `'force'` instructs the `PSPMequi` file to force re-compilation of the model specific file into a `mex` module that can be executed by Matlab. 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 `mex` file. However, if for some unclear reason this automatic recompilation fails, the `'force'` option can be used to initiate re-compilation.

- Option `'debug'`: This option instructs the `PSPMequi` function to turn on debugging flags while compiling the model specific file into a `mex` 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 Matlab 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.

13.2   An example using the `PSPMind` function

To illustrate the use of the `PSPMind` function, I will consider the example as discussed in section 8.5 and shown in box 8.H. 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 (`.mat`) file. As shown in box 8.H the vector of environmental variables for the computed model equilibrium equals `[0.00025335 0.0001336 4.008e-06]`. Computing the individual life history for this particular environmental state with `PSPMind` gives the following result:

```
   >> output = PSPMind('PNAS2002', [0.00025335 0.0001336 4.008e-06], [], {'isort', '1'});

   Building executable PNAS2002ind.mexmaci64 using sources from /Users/andre/programs/PSPM analysis ...



                               Istate[ 0]  Istate[ 1]  Survival        R0  Impact[ 0]  Impact[ 1]  Impact[ 2]  Impact[ 3]
   Pop. # 0 - Bstate  0 - (Final):    1237.24     283.066      1E-09  0.999838   0.0512458   0.0136148   0.0370505    0.624898


       Parameters: [0.1 0.0003 7 27 110 300 9e-06 0.0001 1.5e-05 0.006 0.003 0.01 5000 0.1 0.5 0.01]
```

```
14        Environment: [0.00025335 0.0001336 4.008e-06]
               Pop00: [101x8 double]
```

The individual life history is contained in the structure element `Pop00` and contains the following columns:

```
 2   >> output.Pop00

     ans =
 4
             1           0          7          0           0          0          0          0
 6      0.32214      1.6741     9.7607    0.0063037    0.004973        0          0          0
        0.10259      3.3651     12.521    0.0099518   0.0087971        0          0          0
 8     0.032289      5.0735     15.282    0.011783     0.011195        0          0          0
     <...output lines suppressed in this box...>
10    3.6519e-07      647.2     277.54    0.050977     0.013615    0.037051    0.61773    0.99177
      1.2064e-07     757.96     280.31    0.051156     0.013615    0.037051    0.6225     0.99715
12       1e-09       1237.2     283.07    0.051246     0.013615    0.037051    0.6249     0.99984
```

The columns refer to the individual survival, the values of all individual state variables, the values of the different *cumulative* impacts of the individual and the value of its lifetime reproductive success R_0, respectively. These values are reported at 100 points in the individual life history. Passing the option vector {'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 equidistant subintervals. It is at the boundaries of these subintervals that the life history data are reported. 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`.

Command box 13.B: PSPMind help page

```
   >> help PSPMind
2    PSPMind: Computes the individual life history in a given environment

4    Syntax:

6      output = PSPMind(modelname, environment, parameters, options, 'clean', 'force', 'debug')

8    Arguments:

10     modelname:   (string, required)
                    Basename of the file with model specification. The file
12                  should have extension '.h'. For example, the model 'PNAS2002'
                    is specified in the file 'PNAS2002.h'

14
       environment: (row vector, required)
16                  Vector of length ENVIRON_DIM (set in the model program
                    file), specifying the value of the environmental variables
18                  at which to calculate the individual life history

20     parameters:  (row vector, required, but can be the empty vector [])
                    Vector of length PARAMETER_NR (set in the model program
22                  file), specifying the values for the model parameters to
                    use in the computation. Vectors of other lengths, including
24                  an empty vector will be ignored.

26     options:     (cell array, required, but can be the empty cell array {})
                    Cell array with a pair of an option name and a value (for
28                  example {'isort', '1'}). The only possible option name
                    and its values is:

30
                    'isort',  '<index>': Index of i-state variable to use as
32                                        ruling variable for sorting the
                                          structured populations

34     'clean':     (string. optional argument)
36                  Remove all the result files of the model before the
                    computation

38
       'force':     (string, optional argument)
40                  Force a rebuilding of the model before the computation

42     'debug':     (string, optional argument)
                    Compile the model in verbose mode and with debugging flag set

44
   Output:

46
     The output is a structure with the population state as normally stored in the
48   .mat output file of PSPMdemo, PSPMequi, and PSPMevodyn.
```

Additional information

14

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, used in chapter 4 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, used in chapter 7 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 directory `Test` 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.

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:

1. The number of possible states at birth has to be defined larger than 1.

2. The values of the *i-state* variables at birth have to be defined separately for the different states-at-birth.

3. Not only the number of offspring produced has to be specified, but also the state-at-birth of the offspring has to be specified.

4. 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()` 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.

14.1   Demographic analysis with multiple states-at-birth

14.1.1   Two different offspring body sizes

The file KlanjscekDEB2.h implements a model, in which the life history of the individ-
uals 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 individ-
uals 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 im-
plemented 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 Matlab script
KlanjscekDEB_demo.m 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 box 14.1: Specification of the number of possible states-at-birth

```
    /*
     *===========================================================================
     *  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 the state-at-birth ϕ_2 (BirthStateNr $= 1$) corresponds to the large-sized
offspring with body size $1.3 \cdot V_b$.

Code box 14.2: Specifying the value of all possible states-at-birth

```
    /*
    *=============================================================================
    * 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 `KlanjscekDEB_demo.m` script).

Code box 14.3: Specification of fecundity

```
   /*
    *=============================================================================
    * 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 test script `alldemotests.m`:

```
    >> [data2, desc2] = PSPMdemo('KlanjscekDEB2', [0 1.0 -0.02 0.4 1.0], 'clean', 'force');

    Building executable KlanjscekDEB2demo.mexmaci64 ...

      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
```

```
12      4.00000000E-01  2.03431517E-02

14   #
     # Executing : PSPMdemo('KlanjscekDEB2', [0 1 -0.02 0.4 1], [], {})
16   #
     # Parameter values  :
18   #
     # Food     : 1            Kappa    : 0.8          Kappa_R  : 0.001
20   # Nu       : 0.075        m        : 0.583        g        : 1.286
     # Vb       : 1E-09        Vp       : 1.73E-06     [Em]     : 0.7
22   # ha       : 0.15
     #
24   # Index of bifurcation parameter #1                    : 0
     #
26   #     1:Food    2:PGR[ 0]    3:Tc[ 0]  4:S[ 0][ 0]  5:S[ 0][ 1]  6:S[ 0][ 2]  7:S[ 0][ 3]  .. 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 the output file `KlanjscekDEB2-PGR-0000.mat` containing the stable population states and displaying its contents reveals the additional elements due to the multiple states at births:

```
   >> load('KlanjscekDEB2-PGR-0000.mat', 'State_1_000000E00')
2  >> State_1_000000E00

4  State_1_000000E00 =

6                    BifPars: 1
                  Parameters: [1 0.8 0.001 0.075 0.583 1.286 1e-09 1.73e-06 0.7 0.15]
8                         PGR: 0.69551
       Pop00_StableBirthDist: [0.70973 0.29027]
10         Pop00_BirthStates: [2x4 double]
             Pop00_Bstate00: [100x6 double]
12           Pop00_Bstate01: [100x6 double]
```

The first additional element of the structure 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.

14.1.2   Periodic environments

The file `Medfly_periodic.h` implements a variant of the Medfly model that is discussed in chapter 4, 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, pp. 8-10) and will thus not be presented further here. The Matlab script `Medfly_periodic_demo.m` 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 long time (several hours) 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.

14.2   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 chapters 6-8 and 9-11, 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 Matlab scripts `PNAS2002_5bs_demo.m` and `Indet_growth_5bs_demo.m` carry out the same analysis steps as presented in detail in chapters 8 and 11, 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 box 14.4: Specification of the number of possible states-at-birth

```
    /*
     *===========================================================================
     *  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 thru 4). Of course, for all states-at-birth the age of the individual is set to 0.

Code box 14.5: Specifying the value of all possible states-at-birth

```
   /*
    *===========================================================================
    * 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]` thru `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 Matlab script `PNAS2002_5bs_demo.m` performs the same analysis steps for the PNAS2002 model with 5 states-at-birth, as carried out by the Matlab script `PNAS2002equi_demo.m` 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 Matlab script `Indet_growth_5bs_demo.m` that performs the analysis of the model presented in chapter 10 but with 5 states-at-birth with the results of the original model, the analysis of which was discussed in chapter 11. In both cases the additional states-at-birth hence hardly affect model predictions.

Code box 14.6: Specification of fecundity

```
 1   /*
 2    *==========================================================================
 3    * Specify the fecundity of individuals as a function of the i-state
 4    * variables and the individual's state at birth for all populations in every
 5    * life stage.
 6    *
 7    * The number of offspring produced has to be specified for every possible
 8    * state at birth in the variable 'fecundity[][]'. The first index of this
 9    * variable refers to the number of the structured population, the second
10    * index refers to the number of the birth state.
11    *
12    * Notice that the first index of the variable 'istate[][]' refers to the
13    * number of the structured population, the second index refers to the
14    * number of the individual state variable. The interpretation of the latter
15    * is up to the user.
16    *==========================================================================
17    */
18
19   void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
20                  double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
21                  double *fecundity[POPULATION_NR])
22   {
23     double          fec;
24
25     fecundity[0][0] = 0.0;
26     fecundity[0][1] = 0.0;
27     fecundity[0][2] = 0.0;
28     fecundity[0][3] = 0.0;
29     fecundity[0][4] = 0.0;
30     if (lifestage[0] == 2)
31       {
32         fec = RM*R/(R + RH)*LENGTH*LENGTH;
33         fecundity[0][0] = 0.1*fec;
34         fecundity[0][1] = 0.2*fec;
35         fecundity[0][2] = 0.5*fec;
36         fecundity[0][3] = 0.2*fec;
37         fecundity[0][4] = 0.1*fec;
38       }
39
40     return;
41   }
```

14.3   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 analysed for its population and even community consequences.

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 using the macro constant `REPRODUCTION_INTERVAL`, as for example shown in the command box below.

```
   // The following definition will force the program to consider reproduction pulses
2  #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` in the `Tests` directory 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 14.1, 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 characterising 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:

```
   #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:

```
    void Development(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
 2                  double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                    double development[POPULATION_NR][I_STATE_DIM])
 4  {
      double              dVda, dQda, dHda;
 6    double              Er;

 8    //  Assume growth always occurs
      dVda = max((FOOD*NU*pow(VOLUME, 2.0/3.0) - M*G*VOLUME)/(FOOD + G), 0);
10    dQda = G*EM*(dVda + M*VOLUME);
      dHda = HA*Q/VOLUME;

12
      development[0][0] = 1.0;
14    development[0][1] = dVda;                            // dV/da
      development[0][2] = dQda;                            // dQ/da
16    development[0][3] = dHda;                            // dH/da

18    if (lifestage[0] == 1)                              // Only for adults
        {
20        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);
22        development[0][4] /= EM*(KAPPA*G + FOOD)*VB/KAPPA_R;
        }
24    else
        development[0][4] = 0;

26
      return;
28  }
```

Finally, if reproduction is modelled 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.

```
    void Fecundity(int lifestage[POPULATION_NR], double *istate[POPULATION_NR],
 2                double *birthstate[POPULATION_NR], int BirthStateNr, double E[],
                  double *fecundity[POPULATION_NR])
 4  {
      if (lifestage[0] == 1) // Only for adults
 6      {
          fecundity[0][0] = EGGS;
 8      }
      else
10      fecundity[0][0] = 0;

12    EGGS = 0.0; // Empty the egg buffer

14    return;
    }
```

The demonstration script `KlanjscekDEB_demo` in the `Tests` directory 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`.

16

UNIX command-line usage

Even though in the previous chapters its use from a Matlab or Octave console was discussed, the software itself is a standard C-program. All programs described in previous chapters can hence also be compiled outside Matlab and Octave and used from a command line in, for example, a bash shell under a Unix-based operating system (such as provided by the Terminal.app on Mac OS X). In this chapter I will shortly discuss how to compile and execute the programs within such an interactive bash shell. I will assume that the reader is familiar with the Unix operating system and command line use in bash shells.

The use of the program from a bash command line does not influence the model specification in the `.h` file. Hence, for details of the model implementation I refer to sections 4.1.1 to 4.2.7 for the implementation of a model, which is going to be investigated using demographic analysis, and to the sections 7.1.1 to 7.3.2 for a model that is to be investigated using bifurcation analysis.

16.1 The Makefile and compilation of a program

The distribution includes a Makefile that can be used to build the computational programs. The following box shows the top lines in this Makefile.

```
   # Generic Makefile for the compilation of a continuation problem of the class
 2 # of models that can be analyzed with the generic program files PSPMevodyn.c,
   # PSPMequi.c PSPMind.c or PSPMdemo.c
 4 #
   # Possible targets to make:
 6 #
   #      make <model>demo       (builds demographic executable from .h file)
 8 #      make <model>ind        (builds individual executable from .h file)
   #      make <model>equi       (builds equilibrium executable from .h file)
10 #      make <model>evodyn     (builds evolutionary executable from .h file)
   #
12 #      make clean             (cleans up all target programs)
   #      make cleanoutput       (cleans up all error and output files)
14 #      make allclean          (combines previous two)
   #
16 #
   #=========================================================================
18 # Specify where the library modules are located (ADAPT TO YOUR INSTALLATION)
   #=========================================================================
20
   MODDIR = $(HOME)/programs/PSPManalysis
22
   #=========================================================================
24 # Specify the optional compilation settings and uses of libraries:
   #
26 # CLANG    : Use (1)/Don't use (0) Mac OS X clang compiler (for Mac OS X only)
   # ICC      : Use (1)/Don't use (0) Intel's icc compiler (for Linux only)
28 # OPENMP   : Use (1)/Don't use (0) OpenMP parallelisation
   # MKL      : Use (1)/Don't use (0) Intel's MKL libraries (for Linux only)
30 #
```

```
     # Best settings:
32   #
     # Mac OS X : CLANG = 1, OPENMP = 1 (if clang with -fopenmp support is installed)
34   #            CLANG = 0, OPENMP = 1 (otherwise; gcc with -fopenmp is faster)
     #
36   #=============================================================================

38   CLANG    = 1
     ICC      = 1
40   OPENMP   = 1
     MKL      = 0
```

To adapt this Makefile to your system the definition of the variable `MODDIR` has to be changed. It should point at the directory in which the basic program files such as PSP-Mdemo.c and PSPMequi.c are located.

The variables `CLANG`, `ICC`, `OPENMP` and `MKL` can be set to 0 or 1 and tailor the program to the installed compiler and libraries on your system. On Mac OS system the Makefile will use the *clang* compiler if `CLANG` is set equal to 1, otherwise it invokes the *gcc* compiler. In general the *clang* compiler seems to yield slightly faster code than the *gcc* compiler, but support by the *clang* compilers on Mac OS for the OpenMP parallel programming interface (see below) is limited (except for more recent versions of Mac OS). On Linux systems the Makefile will invoke Intel's *icc* compiler if `ICC` is set equal to 1, otherwise it invokes the *gcc* compiler. Again, the *icc* compiler seems to yield generally somewhat faster code than the *gcc* compiler.

The variable `OPENMP` is relevant on both Mac OS and Linux systems and indicates whether the OpenMP parallel programming interface should be used (`OPENMP` equal to 1) or not (`OPENMP` equal to 0). The use of the OpenMP parallel programming interface is an integral part of the program and requires no further adjustments of the code by the user. Its use yields in particular much faster execution speeds if there are multiple states at birth possible. The speed up by using OpenMP is in these cases substantial.

Finally, the variable `MKL` is only used on Linux. When set to 1, the Makefile links against Intel's MKL libraries as opposed to the standard Lapack and Cblas libraries. This might yield faster code in particular cases, but this speed up is not tested extensively.

Further adaptations to the Makefile are most likely not needed. Note, however, that a number of libraries have to be installed for the programs to compile successfully. In particular, on Mac OS the Makefile assumes that the Lapack and Cblas libraries are installed, as it uses the linking options `-llapack -lcblas -lm` during the final link step of the programs. On Linux system the default linking options are `-llapack -lcblas -llapack_atlas -latlas`, indicating that the Atlas version of the Lapack library is used. Successful compilation requires that the libraries invoked by these linking options are installed on your operating system and can be found by the compiler. However, even though the program uses standard C code and only calls well-established routines from the Lapack and Cblas libraries, the different versions of the Mac OS and Linux operating system differ in the libraries that are installed. This means that the Makefile might not always work on all systems and might have to be adapted if compilation of a program leads to error messages that either of these libraries can not be found.

The Makefile defines a number of targets as is also visible from the command box above. Executing the command `make <model>demo` will build a program for demographic analysis of the model that is specified in the file `<model>.h`. Hence, building the program for the Medfly model discussed as an example in chapters 3 to 5, can be done by executing `make Medflydemo`.

Similarly, executing the commands `make <model>equi`, `make <model>evodyn` and `make <model>ind` will build programs for bifurcation analysis, evolutionary dynamics simulations and individual life history simulations, respectively, of a model that is specified in the file `<model>.h`. Hence, building the program for the PNAS model discussed as an example in chapters 6 to 8, can be done by executing `make PNAS2002equi`.

The remaining targets defined by the Makefile are all cleaning targets. The command `make clean` will clean up all files in the current directory that have been produced during compilation steps. In particular, this step will scan the directory for all the `.h` files in it and will remove all the existing `<model>demo` and `<model>equi` programs, as well as the object modules that were produced during intermediate compilation steps of these programs.

Executing the command `make cleanoutput` will delete all files that have been produced by running either a demographic analysis program or a bifurcation analysis program. In particular, this step will scan the directory for all files with a name `<model>.h`, which will all be considered to be model-specific include files. For each of these files `<model>.h` it will delete all files in the current directory that start with `<model>` and have an extension of `.bif`, `.err`, `.mat`, `.csb` or `.out`. This will remove all output files generated by the computation modules, whether they have been executed from the Matlab command line or from the bash command line.

Finally, the command `make allclean` combines the previous two cleaning steps into one.

16.2 Executing a compiled program

After building a program for demographic analysis, its use from the command-line follows a similar approach as its execution from Matlab, only the syntax is different because all options, initial values for the computation, the step size in the bifurcation parameter and the limits to the parameter range have to be passed as command-line arguments. The syntax can be inspected by executing the compiled program with a single argument `-?`, which prints out a help message:

Command box 16.A: Command-line syntax for demographic analysis

```
     ~/programs/PSPManalysis/Tests: Medflydemo -?
 2
     Usage: Medflydemo [-par1 <index> <initial value> <step> <min. par.1> <max. par.1>] [-isort <index>] [-test]
 4
     Aim:    Computation of the population growth rate and its parameter sensitivities for (multiple) structured populations
 6           possibly as a function of a model parameter

 8   Possible options are:

10   -par1   <index>: Index of the parameter, as a function of which to compute the population growth rate
                     If this option is used <initial value>, <step>, <min. par.1> and <max. par.1> specify
12                   the initial value, stepsize value, minimum and maximum value of this parameter.
                     If not specified, a single computation of the population growth rate and parameter sensitivities
14                   is carried out for the current parameter values.
     -isort  <index>: Index of i-state variable to use as ruling variable for sorting the structured populations.
16   -test          : Perform only a single integration over the life history, reporting dynamics of survival, R0
                      and i-state variables.
18
     Medflydemo, Copyright (C) 2015, Andre M. de Roos, University of Amsterdam
20
     This program comes with ABSOLUTELY NO WARRANTY; without even the implied warranty of
22   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
     License (<http://www.gnu.org/philosophy/why-not-lgpl.html>) for more details
24
```

The help message in the box above shows that executing the computations from the command-line takes the same options as when executing the module from Matlab as

explained in chapter 5. I will hence not discuss in detail the structure of the command-line, as the help message shown above in combination with the explanation about the demographic analysis program in chapter 5 should suffice to infer the correct use of the program.

As an example the box below shows how to execute the demographic analysis from the command-line without any further parameters. This example corresponds to the call to the `PSPMdemo` function shown in Matlab command box 5.A. Comparing the output below with that in box 5.A shows that the two are identical. For further explanations I therefore refer to the explanation of the use of the computational program from Matlab in chapter 5.

Command box 16.B: Demographic analysis call for a single parameter value

```
     ~/programs/PSPManalysis/Tests: Medflydemo
 2   #
     # Executing : Medflydemo
 4   #
     # Parameter values  :
 6   #
     #  Beta0    :  47          Beta1    :  0.04         AJ        :  11
 8   #  Mu0      :  0.00095     Mu1      :  0.0581
     #
10   #   0:PGR[ 0]     1:Tc[ 0]  2:S[ 0][ 0]  3:S[ 0][ 1]  4:S[ 0][ 2]  5:S[ 0][ 3]  6:S[ 0][ 4]
     #
12       0.419057    13.167260     0.001616    -0.164595    -0.031982    -1.526368    -0.011325
```

In a similar vein, the output of an execution of the program to compute the population growth rate over a range of parameters is the same as in Matlab command box 5.B, as shown in the following box:

Command box 16.C: Demographic analysis call for a parameter range

```
     ~/programs/PSPManalysis/Tests: Medflydemo -par1 2 11 0.1 11 20
 2    1.10000E+01     4.19057E-01
      1.11000E+01     4.15884E-01
 4    1.12000E+01     4.12763E-01
     <...output suppressed in this box...>
 6    1.98000E+01     2.53880E-01
      1.99000E+01     2.52772E-01
 8    2.00000E+01     2.51674E-01
```

Notice that the placement of the options and their subsequent values likes `-par1 2` on the command-line is arbitrary, but that the initial value, step size and minimum and maximum parameter value should be specified in this specific order.

Executing a compiled program for bifurcation analysis from the bash command-line (with or without an option `-?`) also prints out a help message that reveals the syntax to be used for such programs, as shown in the following box:

```
     ~/programs/PSPManalysis/Tests: PNAS2002equi
 2
     Usage: PNAS2002equi [<options>] <Type> <Initial values> <max. parameter step> <index par.1> <min. par.1> <max. par.1> ...
 4                            ... [<index par.2> <min. par.2> <max. par.2> ... <index par.n> <min. par.n> <max. par.n>]

 6   Aim:    Continuation of trivial or non-trivial equilibria, transcritical and saddle-node bifurcations
                 of structured populations as well as transcritical bifurcation in one of its environment variables
 8                 and evolutionary continuation as a function of one or two parameters

10   <Type>: Type of curve computation to be performed, either BP, EQ, LP, BPE, ESS or PIP

12   Possible options are:

14   -envBP   <index> : Index of environment variable, of which to continue the transcritical bifurcation
     -popBP   <index> : Index of structured population, of which to continue the transcritical bifurcation
16   -popEVO  <index> : Index of structured population, for which to compute selection gradient or perform ESS or PIP continuation
     -parEVO  <index> : Index of parameter to compute selection gradient for during EQ continuation
18   -envZE   <index> : Index of environment variable in trivial equilibrium (can be used multiple times)
     -popZE   <index> : Index of structured population in trivial equilibrium (can be used multiple times)
```

```
20    -evoPars <number>: Number of life history parameters of structured population at their ESS value
      -isort   <index> : Index of i-state variable to use as ruling variable for sorting the structured populations
22    -noBP             : Do not check for branching points while computing equilibrium curves
      -noLP             : Do not check for limit points while computing equilibrium curves
24    -single           : Only compute the first point of the solution curve, do not continue the curve
      -test             : Perform only a single integration over the life history, reporting dynamics of survival, R0,
26                        i-state and interaction variables

28    The value for -evoPars defaults to 0 unless set on the command-line
      The values for -envBP, -popBP, -popEVO, -parEVO, -envZE and -popZE are undefined unless set on the command-line
30
      ./PNAS2002equi, Copyright (C) 2015, Andre M. de Roos, University of Amsterdam
32
      This program comes with ABSOLUTELY NO WARRANTY; without even the implied warranty of
34    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
      License (<http://www.gnu.org/philosophy/why-not-lgpl.html>) for more details
```

This help message becomes more specific once the type of computation that has to be carried out is specified, as illustrated in the code box below:

```
      ~/programs/PSPManalysis/Tests: PNAS2002equi BPE
2
      Set the environmental variable index for BPE continuation (0 <= i < 3) using the 'envBP' option!
4
      Usage: PNAS2002equi [<options>] BPE Par.1 E[0] E[1] E[2] b[0] Par.2 <max. parameter step> <index par.1> <min. par.1> ...
6                                                  ... <max. par.1> <index par.2> <min. par.2> <max. par.2>

8     Aim:    Continuation of a transcritical bifurcation in one of the environment variables
              of a structured population as a function of two parameters.
10            Only works if the dynamics of the environment variable
              is linear in the variable itself
12
      Possible options are:
14
      -envBP   <index> : Index of environment variable, of which to continue the transcritical bifurcation
16    -popBP   <index> : Index of structured population, of which to continue the transcritical bifurcation
      -popEVO  <index> : Index of structured population, for which to compute selection gradient or perform ESS or PIP continuation
18    -parEVO  <index> : Index of parameter to compute selection gradient for during EQ continuation
      -envZE   <index> : Index of environment variable in trivial equilibrium (can be used multiple times)
20    -popZE   <index> : Index of structured population in trivial equilibrium (can be used multiple times)
      -evoPars <number>: Number of life history parameters of structured population at their ESS value
22    -isort   <index> : Index of i-state variable to use as ruling variable for sorting the structured populations
      -noBP             : Do not check for branching points while computing equilibrium curves
24    -noLP             : Do not check for limit points while computing equilibrium curves
      -single           : Only compute the first point of the solution curve, do not continue the curve
26    -test             : Perform only a single integration over the life history, reporting dynamics of survival, R0,
                          i-state and interaction variables
28
      The value for -evoPars defaults to 0 unless set on the command-line
30    The values for -envBP, -popBP, -popEVO, -parEVO, -envZE and -popZE are undefined unless set on the command-line
32    PNAS2002equi, Copyright (C) 2015, Andre M. de Roos, University of Amsterdam
34    This program comes with ABSOLUTELY NO WARRANTY; without even the implied warranty of
      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
36    License (<http://www.gnu.org/philosophy/why-not-lgpl.html>) for more details
```

The help message in the boxes above show that executing the computations from the command-line is quite similar to executing the module from Matlab as explained in section 8.2. I will hence not discuss in detail the structure of the command-line, the meaning of the options and the required input variables for the program as the help message shown above in combination with the explanation about the bifurcation analysis program in section 8.2 should suffice to infer the correct use of the program. One thing to notice, though, is that in all calls to a bifurcation analysis program the option `-par1 <index>` has to be specified. If this option is not specified on the command-line the program exists with an appropriate error message. The placement of the options and their subsequent values on the command-line is again arbitrary, but all other input quantities, such as the initial values of all variables, the step size along the curve and minimum and maximum parameter values should be specified in the specific order as shown in the help message.

As an example the box below shows how to execute the same calculation for the PNAS model as illustrated in Matlab command box 8.B.

Command box 16.D: Command-line example of a bifurcation analysis program

```
      ~/programs/PSPManalysis/Tests: PNAS2002equi -par1 1 -popZE 0 -envZE 1 -envZE 2 EQ 1.0E-06 1.0E-06 0.5 0 4E-4
 2

      1.00000E-06      1.00000E-06
 4    1.03536E-06      1.03536E-06
      1.07071E-06      1.07071E-06
 6    <...output suppressed in this box...>
      9.18697E-06      9.18697E-06
 8    8.85690E-06      8.85690E-06    ****  BP #0   ****
      9.54052E-06      9.54052E-06
10    <...output suppressed in this box...>
      3.33749E-04      3.33749E-04
12    3.69104E-04      3.69104E-04
      4.04460E-04      4.04460E-04

14
```

The output in the box above is identical to the output shown in box 8.B. For further details I therefore refer to the explanation of the use of the computational program from Matlab in sections 8.2 and 8.4.

Furthermore, in the directory `Tests` of the package a shell script `allequitests.sh` is provided that illustrates a large number of computations using command-line execution. Executing `. allequitests.sh` from the command line not only tests that the package functions correctly, but also illustrates how to use the program for different types of computations and with a variety of options.

16.3   Output files

When executing a compiled program from the command line the same output files are produced as discussed in sections 5.2 and 8.5, except for the files with an extension `.mat` that are Matlab-specific files. Instead, the information about the stable population states are written to a binary file with extension `.csb`. Hence, the call to the program `Medflydemo` as shown in command box 16.C generates the output files `Medfly-PGR-0001.err`, `Medfly-PGR-0001.out` and `Medfly-PGR-0001.csb`, as opposed to the files `Medfly-PGR-0001.err`, `Medfly-PGR-0001.out` and `Medfly-PGR-0001.mat` produced by the corresponding Matlab command in command box 5.B (the numeric identifier `0001` results because the Medfly program was first executed without parameters, which produces output files with identifier `0000`). Similarly, the call to the program `PNAS2002equi` in box 16.D above produces output files `PNAS2002-EQ-0000.bif`, `PNAS2002-EQ-0000.err`, `PNAS2002-EQ-0000.out` and `PNAS2002-EQ-0000.csb`, as opposed to the files `PNAS2002-EQ-0000.bif`, `PNAS2002-EQ-0000.err`, `PNAS2002-EQ-0000.out` and `PNAS2002-EQ-0000.mat` produced by the corresponding Matlab command in command box 8.B.

The files with extensions `.bif`, `.err` and `.out` will not be discussed further here. Refer to the sections 5.2 and 8.5 for details about their contents. The files with extension `.csb`, which from hereon will be referred to as CSB file, are binary files with a specific structure. The format of these files is inherited from the software package called 'EBTtool' that I developed to perform numerical integrations of PSPMs and which is freely available from my webpage: http://staff.fnwi.uva.nl/a.m.deroos/EBT/Software/index.html. The EBTtool program contains a graphical interface that allows you to graphically inspect the population states stored in the CSB file, browse through the various states in a file and also output the population states to a text file. It hence provides an easy approach for graphical investigation of the stable population states at different values of the bifurcation parameter.

For those users that do not want to install the EBTtool program, however, a utility program is include, called `csb2txt.c` that will read the binary CSB file and translate it to plain text. The source file `csb2txt.c` can be found in the same directory as where the files `PSPMdemo.c` and `PSPMequi.c` are located. This source file can be compiled into an executable program by issuing the command `make csb2txt` in the directory with the source file `csb2txt.c`. The compilation should not pose any major problems as the program is written in fairly standard `C`.

Executing the `csb2txt` program by itself generates a help message that explains its usage as shown below:

```
  ~/programs/PSPManalysis: csb2txt
2
  Usage: csb2txt [-f "printf() format string"] <CSB filename> [parameter value]
4
```

When the program is called with the name of a CSB file (with or without the extension `.csb` does not matter), it will write the information in the entire CSB file to the terminal (unless of course this output is piped into an output file). Alternatively, the name of the CSB file can be followed by a particular parameter value, in which case the program searches in the CSB file for the population state that is closest to the specified value and writes this single population state to the terminal.

Hence, the stable population distribution for the third population state (for parameter value 11.1908) stored in `Medfly-PGR-0001.csb`, which results from the calculations illustrated in box 16.B above, can be inspected with the following call to the `csb2txt` program:

```
   ~/programs/PSPManalysis/Tests: ../csb2txt Medfly-PGR-0001 1.12000E+01
2  # Population growth rates for parameter value 11.2
   11.2000000000    0.4127626846
4
   # State of population #0 for parameter value 11.2
6  1.0000000000    0.0000000000    1.0000000000
   0.8129899031    0.5004188126    1.2300275762
8  0.6609431768    1.0008376252    1.5129893689
   <...output suppressed in this box...>
10 1.536459E-09    48.5406248235   8.9188991230
   1.239653E-09    49.0410436361   5.2194771968
12 1.000000E-09    49.5413326488   0.0000000000
```

A large number of intermediate lines are suppressed in the output of the command, as it would take up too much space. The population distribution displayed is, however, identical to the population distribution that is presented as an example in section 5.2 when discussing the contents of the `Medfly-PGR-0001.mat` file. The first line of output specifies the value of the bifurcation parameter as the first entry on the line and the calculated population growth rate value as the second entry. The following 3 columns represent the stable population density, the corresponding age as well as the reproductive value of the individuals at that age, respectively.

Similarly, the last population state stored in the file `PNAS2002-EQ-0000.csb` can be inspected by executing the following command, which uses the last parameter value of the computations `4.04758E-04` for selecting the state:

```
     ~/programs/PSPManalysis/Tests: ../csb2txt PNAS2002-EQ-0000 4.04460E-04
2    # Environment variables for parameter value 0.00040446
     0.0004044597     0.0004044597     0.0000000000     0.0000000000
4
     # State of population #0 for parameter value 0.00040446
6    0.0000000000     10.0040106825    23.2738732599
     0.0000000000     30.7272765409    54.3743905369
8    0.0000000000     51.4505423942    81.8386322000
     <...output suppressed in this box...>
10   0.0000000000     2020.1607969467 289.2703746568
     0.0000000000     2040.8840627837 289.2705544476
12   0.0000000000     2061.6073286205 289.2707132173
```

As before, a large number of intermediate lines are suppressed in the output of the command for brevity.

The layout of the stable population distribution is always the same, irrespective of the type of bifurcation computation that has been carried out. The first line of the output contains the value of the bifurcation parameter as first entry and the equilibrium values of all environment variables as subsequent entries. After an empty line, the stable population distribution is printed. Each line of this distribution corresponds to a cohort of individuals with the first entry on a line specifying the number of individuals in that cohort and the subsequent entries on the line the average values of the individual state variables of the individuals in the cohort, in the order as adopted in the model specification file. Hence, for the PNAS model the second and third entry on a row specifies the average age and length, respectively, of the individuals in a cohort (see code box 7.3). The example of the stable population distribution shown in the box above is of course rather special, because the computations pertain to a curve with a zero equilibrium state of the structured consumer population. For this reason, the cohort density in the first entry on each row is consistently 0. This would, however, be different in the more general case of an equilibrium curve with a non-zero equilibrium state of the consumer.

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 boxes 4.1 and 7.1.

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
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, either in a Matlab file with `.mat` extension or 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.

Setting name	Default value	Interpretation
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
ODESOLVE_MIN_STEP	10^{-8}	Smallest possible step size in the numerical integration of the ODEs
ODESOLVE_MAX_STEP	10.0	Largest possible step size in the numerical integration of the ODEs
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.
ODESOLVE_ABS_ERR	10^{-10}	Absolute error in the numerical integration of the ODEs
ODESOLVE_REL_ERR	10^{-8}	Relative error in the numerical integration of the ODEs
ODESOLVE_FUNC_TOL	10^{-8}	Threshold value determining whether a stopping event in the numerical integration routine has been detected
JACOBIAN_MIN_STEP	10^{-7}	Absolute minimum change in variable when computing Jacobian matrix
JACOBIAN_STEP	10^{-4}	Relative change in variable when computing Jacobian matrix
JACOBIAN_UPDATES	5	Number of Newton adjustments before the Jacobian matrix is computed anew

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 *et al.* (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:

$$\frac{\partial n(t,s)}{\partial t} + \frac{\partial \left(g(s,R)n(t,s) \right)}{\partial s} = -\mu(s,R)\, n(t,s) \tag{18.1}$$

$$g(s_b,R)\, n(t,s_b) = \int_{s_b}^{s_m} \beta(s,R)\, n(t,s)\, ds \tag{18.2}$$

$$\frac{dR}{dt} = G(R) - \int_{s_b}^{s_m} \gamma(s,R)\, n(t,s)\, ds \tag{18.3}$$

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:

$$b(t) = \int_0^\infty \beta(s(t,a,R_t),R(t))\, \mathcal{F}(t,a,R_t)\, b(t-a)\, da \tag{18.4}$$

$$\frac{dR}{dt} = G(R(t)) - \int_0^\infty \gamma(s(t,a,R_t),R(t))\, \mathcal{F}(t,a,R_t)\, b(t-a)\, da \tag{18.5}$$

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:

$$s(t, a, R_t) \;=\; s^0 \;+\; \int_0^a g(s(t - \alpha, \alpha, R_{t-\alpha}), R(t - \alpha))\, d\alpha \tag{18.6}$$

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:

$$\mathcal{F}(t, a, R_t) \;=\; \exp\left(-\int_0^a \mu(s(t - \alpha, \alpha, R_{t-\alpha}), R(t - \alpha))\, d\alpha\right) \tag{18.7}$$

Figure 18.1 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.
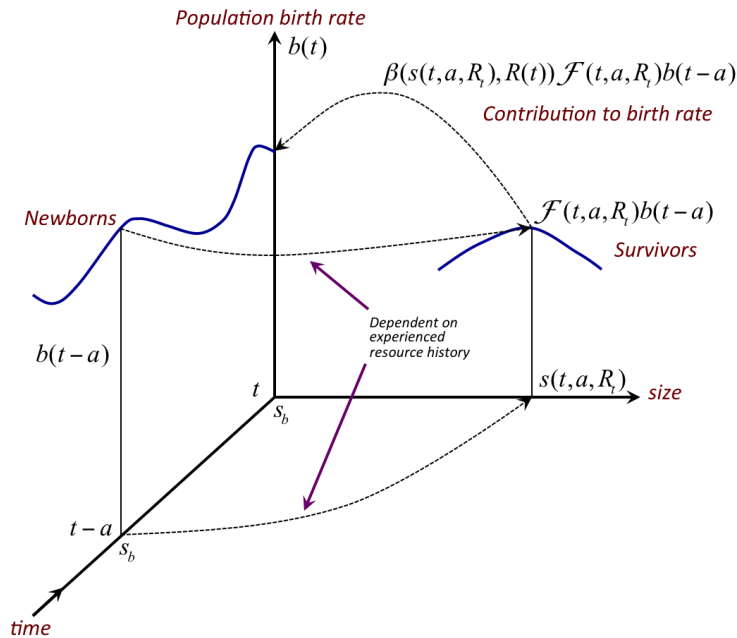

Figure 18.1: Schematic representation of the integro-differential equation system for the size-structured consumer-resource model, showing how the population birth rate at time $t - a$ contributes to the birth rate at time t through consumers of age a that have grown during their life from their size 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.

18.1   The system of equations determining the population growth rate

For demographic analysis of a linear PSPM only the integral equation 18.4 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 18.4 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:

$$\int_0^\infty e^{-ra} \beta(\boldsymbol{\chi}(a)) \, \mathcal{F}(a) \, da \; = \; 1 \tag{18.8}$$

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 18.8 can then be expressed as:

$$H(\infty, r) \; = \; 1 \tag{18.9}$$

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 differentation 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:

$$\frac{d\mathcal{F}}{da} \;=\; -\mu(\boldsymbol{\chi}(a))\,\mathcal{F}(a), \qquad \mathcal{F}(0) \;=\; 1$$

Similarly, differentiate $H(a, r)$ with respect to a and applying Leibniz rule yields:

$$\frac{dH}{da} \;=\; e^{-ra}\,\beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a), \qquad H(0) \;=\; 0$$

The value of $H(\infty, r)$ can hence be calculated by (numerical) integration of the ODEs:

$$\begin{cases} \dfrac{d\boldsymbol{\chi}}{da} \;=\; g(\boldsymbol{\chi}), & \boldsymbol{\chi}(0) \;=\; \boldsymbol{\chi}_b \\[2mm] \dfrac{d\mathcal{F}}{da} \;=\; -\mu(\boldsymbol{\chi}(a))\,\mathcal{F}(a), & \mathcal{F}(0) \;=\; 1 \\[2mm] \dfrac{dH}{da} \;=\; e^{-ra}\,\beta(\boldsymbol{\chi}(a))\,\mathcal{F}(a), & H(0) \;=\; 0 \end{cases}$$

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 18.9 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.

18.2 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:

$$\frac{dI}{da} \;=\; \gamma(\boldsymbol{\chi}(a, \tilde{E}), \tilde{E}) \, \mathcal{F}(a, \tilde{E}), \qquad I(0) \;=\; 0$$

The equilibrium of a nonlinear structured population model is therefore determined by the system of equations:

$$H(\infty, \tilde{E}) \;=\; 1$$
$$\tilde{b} I(\infty, \tilde{E}) \;=\; G(\tilde{E})$$

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:

$$\begin{cases} \dfrac{d\boldsymbol{\chi}}{da} \;=\; g(\boldsymbol{\chi}(a, \tilde{E}), \tilde{E}), & \boldsymbol{\chi}(0, \tilde{E}) \;=\; \boldsymbol{\chi}_b \\[2mm] \dfrac{d\mathcal{F}}{da} \;=\; -\mu(\boldsymbol{\chi}(a, \tilde{E}), \tilde{E}) \, \mathcal{F}(a, \tilde{E}), & \mathcal{F}(0, \tilde{E}) \;=\; 1 \\[2mm] \dfrac{dH}{da} \;=\; \beta(\boldsymbol{\chi}(a, \tilde{E}), \tilde{E}) \, \mathcal{F}(a, \tilde{E}), & H(0, \tilde{E}) \;=\; 0 \\[2mm] \dfrac{dI}{da} \;=\; \gamma(\boldsymbol{\chi}(a, \tilde{E}), \tilde{E}) \, \mathcal{F}(a, \tilde{E}), & I(0, \tilde{E}) \;=\; 0 \end{cases}$$

18.3 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 chapter 9. 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 chapters 9 to 11 relies on the computation of these partial derivatives, which are computed numerically as pointed out above.

Bibliography

De Roos, A.M., 1997. A gentle introduction to physiologically structured population models. In: S. Tuljapurkar & H. Caswell (Eds.), *Structured population models in marine, terrestrial and freshwater systems*, pp. 119–204. Chapman-Hall, New York.

De Roos, A.M., 2008. Demographic analysis of continuous-time life-history models. Ecology Letters 11: 1–15.

De Roos, A.M. & L. Persson, 2002. Size-dependent life-history traits promote catastrophic collapses of top predators. Proceedings of the National Academy of Sciences 99 (20): 12907–12912.

De Roos, A.M. & L. Persson, 2013. *Population and Community Ecology of Ontogenetic Development*, Monographs in Population Biology 51. Princeton University Press, Princeton

Dieckmann, U. & R. Law, 1996. The dynamical theory of coevolution: A derivation from stochastic ecological processes Journal of Mathematical Biology 34(5-6): 579–612.

Dieckmann, U., 1997. Can adaptive dynamics invade? Trends in Ecology & Evolution 12(4): 128–131.

Diekmann, O., M. Gyllenberg & J.A.J. Metz, 2003. Steady-State Analysis of Structured Population Models. Theoretical Population Biology 63 (4): 309–338.

Geritz, S.A.H., E. Kisdi, G. Meszéna, & 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 & C. Rueffler, 2015. Mutual invadability near evolutionarily singular strategies for multivariate traits, with special reference to the strongly convergence stable case. Journal of Mathematical Biology, doi:10.1007/s00285-015-0944-6.

Hairer, E., S.P. Norsett & G. Wanner, 1993. Solving Ordinary Differential Equations I: Nonstiff Problems. Springer-Verlag, Berlin.

Kirkilionis, M.A., O. Diekmann, B. Lisser, M. Nool, B. Sommeijer, & A.M. De Roos, 2001. Numerical continuation of equilibria of physiologically structured population models. I. Theory. Mathematical Models & Methods in Applied Sciences 11 (6): 1101–1127.

Kuznetsov, Y.A., 1995. Elements of Applied Bifurcation Theory. Springer-Verlag, Heidelberg.

Leimar, O., 2005. The evolution of phenotypic polymorphism: Randomized strategies versus evolutionary branching. American Naturalist 165(6): 669–681.

Metz, J.A.J. & O Diekmann, 1986. *The dynamics of physiologically structured populations*, Lecture Notes in Biomathematics 68. Springer-Verlag, Heidelberg.

Metz, J.A.J., S.A.H. Geritz, G. Meszéna, F.J.A. Jacobs, & J.S. van Heerwaarden. 1996. Adaptive Dynamics, a geometrical study of the consequences of nearly faithful reproduction. In: S.J. van Strien & S.M. Verduyn-Lunel (Eds.), *Stochastic and spatial structures of dynamical systems*, pp. 183–231, KNAW Verhandelingen, Amsterdam.