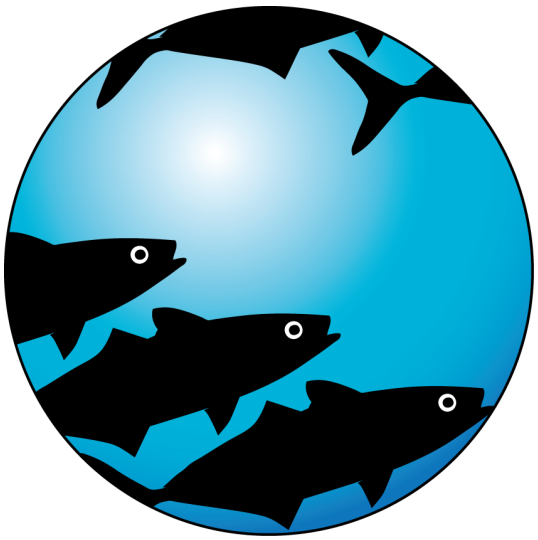
Spatial Population Model User Manual

Alistair Dunn, Scott Rasmussen, and Sophie Mormede



NIWA Technical Report 138
ISSN 1174-2631
2015

Spatial Population Model User Manual (modified: 2015-11-19)
for use with `SPM` v

Published by NIWA
Wellington
2015

Enquiries to:
Science Communication, NIWA
Private Bag 14901, Wellington, New Zealand

ISSN 1174-2631

Citation: Dunn, A.; Rasmussen, S.; Mormede, S. (2015). Spatial Population Model User Manual, `SPM` v
. National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 138*. 208 p.

Contents

1. Introduction

SPM (Spatial Population Model) is a generalised spatially explicit age-structured population dynamics and movement model. SPM can model population dynamics and movement parameters for an age-structured population using a range of observations, including tagging, relative abundance, and age frequency data. SPM implements an age-structured population within an arbitrary shaped spatial structure, which can have user defined categories (e.g., immature, mature, male, female, etc.), and age range.

This manual describes how to use SPM, including how to run SPM, how to set up an input configuration file. Further, we describe the population dynamics and estimation methods, and describe how to specify and interpret output.

1.1. Version

This document (last modified 2015-11-19) describes SPM v

. The SPM version number is suffixed with a date/time (yyyy-mm-dd) and revision number, giving the revision control system UTC date and revision number for the most recent modification of the source files. User manual updates will usually be issued for each minor version or date release of SPM, and can be obtained, on request, from the authors.

1.2. Citing SPM

A suitable reference for SPM and this document is:

Dunn, A.; Rasmussen, S.; Mormede, S. (2015). Spatial Population Model User Manual, SPM v

. National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 138*. 208 p.

1.3. Software license

1.4. System requirements

SPM is available for most IBM compatible machines running 64-bit Linux and Microsoft Windows operating systems.

Several of SPMs tasks are highly computer intensive and a fast processor is recommended. Depending on the model implemented, some of SPMs tasks can take a considerable amount of time (minutes to hours), and in extreme cases can even take several days to estimate a model fit. Some of SPMs tasks can be multi-threaded, and hence multi-core machines may perform some tasks quicker than single core processors.

The program itself requires only a few megabytes of hard-disk space but output files can consume large amounts of disk space. Depending on number and type of user output requests, the output could range from a few hundred kilobytes to several hundred megabytes. When estimating model

fits, several hundred megabytes of RAM may be required, depending on the spatial size of the model, number of categories, and complexity of processes and observations. For extremely large models, several gigabytes of RAM may be required.

1.5. Necessary files

For both 64-bit Linux and Microsoft Windows, only the binary file `spm` or `spm.exe` is required to run `SPM`. No other software is required. We do not compile a version for 32-bit operating systems.

`SPM` offers little in the way of post-processing of the output, and a package available that allows tabulation and graphing of model outputs is recommended. We suggest software such as Microsoft Excel, S-Plus, or **R** (R Development Core Team 2007). To assist in the post processing of `SPM` output, we provide the `spm` **R** package for importing the `SPM` output into **R** (see Section 14).

1.6. Getting help

`SPM` is distributed as unsupported software, however we would appreciate being notified of any problems or errors in `SPM`. See Section 15.2 for how to report errors to the authors. Further information about `SPM` can be obtained by contacting the authors.

1.7. Technical details

`SPM` was compiled on Linux using `gcc`, the C/C++ compiler developed by the GNU Project. The 64-bit Linux version was compiled using `gcc` version 4.8.2. Note that `SPM` is not supported for Linux kernel versions prior to 2.6. The Microsoft Windows version was compiled using Mingw32 `gcc` (tdm64-1) 4.9.2. The Microsoft Windows installer was built using the Nullsoft Scriptable Install System.

`SPM` uses two minimisers — the first is closely based on the main algorithm of Dennis Jr and Schnabel (1996), and which which uses finite difference gradients, and the second is an implementation of the differential evolution solver (Storn and Price, 1995), and based on code by Lester E. Godwin of PushCorp, Inc.

The random number generator used by `SPM` uses an implementation of the Mersenne twister random number generator (Matsumoto and Nishimura, 1998). This, the command line functionality, matrix operations, and a number of other functions use the BOOST C++ library (Version 1.54.0).

Note that the output from `SPM` may differ slightly on the different platforms due to different precision arithmetic or other platform dependent implementation issues. The source code for `SPM` is available either as a part of the installation, or on request from the authors.

Unit tests of the underlying `SPM` code are carried out at build time, using the BOOST unit testing framework. The unit test framework aims to cover the key functions and processes within the `SPM` code base. The unit test code for `SPM` is available as a part of the underlying source code. You can view a summary of the unit tests and validate the version of `SPM` you are using has passed these by running

```
spm_unittests --report_level=detailed
```

from the command line.

2. Model overview

2.1. Introduction

The Spatial Population Model (SPM) is a generalised spatially explicit age-structured population dynamics and movement model. It allows the implementation of age-structured population models suitable for the simulation and estimation of parameters in models with a large number of areas. It implements a statistical catch-at-age population dynamics and movement model, using a discrete time-step state-space model that represents a cohort-based population age structure in a spatially explicit manner.

SPM is run from the console window on Microsoft Windows or from a terminal window on Linux. SPM gets its information from input data files, the main one of which is the *input configuration file*. Commands and subcommands in the input configuration file are used to define the model structure, provide observations, define parameters, and define the outputs (reports) for SPM. Command line switches tell SPM the run mode and where to direct its output. See Section 3 for the details.

The basic structure of an SPM model is a set of spatial cells, each of which contains a population. We define the model in terms of the *state*. The state consists of two parts, the *partition*, and any *derived quantities* or *derived quantities by cell*. The state will typically change one or more times in every *time-step* of every year, depending on the *processes* defined for each model.

The partition is a representation of the population at an instance in time, and is a matrix of the numbers of individuals within each spatial cell, age, and category. A derived quantity is a cumulative summary of the partition (over all cells) at some point in time. A derived quantity by cell is a cumulative summary of the partition in each of the cells at some point in time. Unlike the partition (which is updated as each new process is applied), each derived quantity records a single value for each year of the model run, and each derived quantity by cell records a layer of values for each year of the model run. Hence, derived quantities build up a vector of values over the model run years. For example, the total number of individuals in a category labelled mature at some point in the annual cycle may be a derived quantity and the total number of individuals in a category labelled mature in each cell of the model at some point in the annual cycle may be a derived quantity by cell. The state is the combination of the partition and any derived quantities or derived quantities by cell at some instance in time. Changes to the state occur by the application of processes. Additions to the vectors of derived quantities occur when a model is requested to add a value to each derived quantity vector.

Running of the model consists of two main parts — first the model state is initialised for a number of iterations (years), then the model runs over a range of predefined years.

The application of processes within each year is controlled by the *annual cycle*. This defines what processes happen in each model year, and in what sequence. Initialisation can be phased, and for each phase, the user need to define the processes that occur in each year, and the order in which they are applied.

For the run years, each year is split up into one or more time-steps (with at least one process occurring in each time-step). You can think of each time-step as representing a particular part of the calendar year, or you can just treat them as an abstract sequence of events.

The division of the year into an arbitrary number of time-steps allows the user to specify the exact order in which processes occur and when observations are evaluated. The user specifies the time-steps, their order, and the processes within each time-step. If more than one process occurs in the same time-step, then the occur in the order that they are specified. Observations are always evaluated at the end of the time-step in which they occur. Hence, time-steps can be used to break processes into groups, and assist in defining the timing of the observations within the annual cycle.

An SPM model can be parametrised by both population processes (for example, ageing, recruitment, and mortality) and movement processes. Movement is parameterised by either adjacent cell movements, between cell migrations, or by global movements as a function of known attributes at each spatial location (termed preference functions — see later). SPM is designed to be flexible and to allow for the estimation of both population and movement parameters from local or aggregated spatially explicit observations.

The population structure of SPM follows the usual population modelling conventions and is similar to those implemented in other population models, for example CASAL (Bull et al., 2012). The model records the numbers of individuals by age and category (e.g., male, female), as well as the locations of these cohorts within a spatial grid. In general, cohorts are added via a recruitment event, are aged annually, and are removed from the population via various forms of mortality. The population is assumed to be closed (i.e., no immigration or emigration from the modelled area)

A model is implemented in SPM using an input configuration file, which is a complete description of the model structure (i.e., spatial and population processes), observations, estimation methods, and reports (outputs) requested. SPM runs from a console window on Microsoft Windows or from a text terminal on Linux. A model can be either *run*, estimable parameters can be *estimated* or *profiled*, *MCMC* distributions calculated, and these estimates can be used by SPM as parameters of an operating model to *simulate* observations.

A model in SPM is specified by an input configuration file, and comprises of four main components. These are the population section (model structure, population and spatial dynamics, etc.), the estimation section (methods of estimation and the parameters to be estimated), the observation section (observational data and associated likelihoods), and the report section (printouts and reports from the model). The input configuration file completely describes a model implemented in SPM. See Sections 8, 9, 10, and 11 for details and specification of SPMs command and subcommand syntax within the input configuration file.

2.2. The population section

The population section (Section 4) defines the model of the movement and population dynamics. It describes the model structure (both the spatial and population structure), initialisation and run years (model period), population and movement processes (for example, recruitment, migration, and mortality), layers (the known attributes of each spatial cell), selectivities, and key population parameters.

2.3. The estimation section

The estimation section (Section 5) specifies the parameters to be estimated, estimation methods, penalties and priors. Estimation is based on an objective function (e.g., negative log posterior). Depending on the run mode, the estimation section is used to specify the methods for finding a point estimate (i.e., the set of parameter values that minimizes the objective function), doing profiles, or MCMC methods and options, etc.

The estimation section specifies the parameters to be estimated within each model run and the estimation methods. The estimation section specifies the choice of estimation method, which model parameters are to be estimated, priors, starting values, and minimiser control values.

Penalties and priors act as constraints on the estimation. They can either encourage or discourage (depending on the specific implementation) parameter estimates that are ‘near’ some value, and hence influence the estimation process. For example, a penalty can be included in the objective

function to discourage parameter estimates that lead to models where the recorded mortality was unable to be fully taken.

2.4. The observation section

Types of observations, their values, and the associated error structures are defined in the observation section (Section 6). Observations are data which allow us to make inferences about unknown parameters. The observation section specifies the observations, their errors, likelihoods, and when the observations occur. Examples include relative or absolute abundance indices, proportions-at-age frequencies, etc. Estimation uses the observations to find values for each of the estimated parameters so that each observation is 'close' (in some mathematical sense) to a corresponding expected value.

2.5. The report section

The report section (Section 7) specifies the model outputs. It defines the quantities and model summaries to be output to external files or to the standard output. While `SPM` will provide informational messages to the screen, the `SPM` will only produce model estimates, population states, and other data as requested by the report section. Note that if no reports are specified, then no output will be produced.

3. Running SPM

SPM is run from the console window (i.e., the DOS command line) on Microsoft Windows or from a terminal window on Linux. SPM gets its information from input data files, the key one of which is the input configuration file.

The input configuration file is compulsory and defines the model structure, processes, observations, parameters (both the fixed parameters and the parameters to be estimated), and the reports (outputs) requested. The following sections describe how to construct the SPM configuration file. By convention, the name of the input configuration file ends with the suffix .spm, however, any file name is acceptable.

Other input files can, in some circumstances, be supplied to define the starting point for an estimation or as a point estimate from which to simulate observations.

Simple command line arguments are used to determine the actions or *tasks* of SPM, i.e., to run a model with a set of parameter values, estimate parameter values (either point estimates or MCMC), project quantities into the future, simulate observations, etc,. Hence, the *command line arguments* define the *task*. For example, -r is the *run*, -e is the *estimation*, and -m is the *MCMC* task. The *command line arguments* are described in Section 3.4.

3.1. Using SPM

To use SPM, open a console (i.e., the command prompt) window (Microsoft Windows) or a terminal window (Linux). Navigate to a directory of your choice, where your input configuration files are located. Then type spm with any arguments (see Section 3.4 for the the list of possible arguments). SPM will print output to the screen and return you to the command prompt when it completes its task. Note that the SPM executable (binary) must be either in the directory where you run it or somewhere in your PATH. Note that an automated installer is available for SPM on Microsoft Windows. If you use the installer, then it will give you the option of modifying your PATH for you (as well a a number of other options to make using the program a little easier). Otherwise, see your operating system documentation for help on identifying or modifying your PATH.

3.2. The input configuration file

The input configuration file is made up of four broad sections; the description of the population structure and parameters (the population section), the estimation methods and variables (the estimation section), the observations and their associated likelihoods (the observation section), and the outputs and reports that SPM will return (the report section).

The input configuration file is a plain text file, and is made up of a number of commands (each with subcommands) which specify various options for each of these components. Some of the commands are compulsory, and others are required to specify certain options. Commands always begin with an @ character, with some commands also requiring a label.

Subcommands follow the command, with each subcommand having an argument. Some subcommands have default arguments, but many don't and must be specified. Arguments may be strings, numbers, or vectors of strings or numbers. The type of argument is always specific to the subcommand. The order of subcommands or commands in a file does not matter, except that the subcommands for each command must always follow the associated command and occur before the next command.

For example, to request a report on the state of the partition, use the `@report` with an arbitrary label (i.e., myPartitionReport) that will allow you to identify it in the subsequent output file, and with the appropriate subcommands to specify the year and time-step, use the syntax;

```
@report partition
type myPartitionReport
time_step step_two
years 2007
```

The command and subcommand definitions in the input configuration file can be extensive (especially when you have a model with a large spatial structures that has many layers and/or observations), and can result in a input configuration file that is long and difficult to navigate. To aid readability and flexibility, we can use the input configuration file command `@include` *`file`*. The command causes an external file, *`file`*, to be read and processed, exactly as if its contents had been inserted in the main input configuration file at that point. The file name must be a complete file name with extension, but can use either a relative or absolute path as part of its name. Note that included files can also contain `@include` commands — but be careful that you do not set up a recursive state. See Section 12 for more detail.

3.3. Redirecting standard output

`SPM` uses the standard output stream `standard output` to display run-time information. The standard error stream is used by `SPM` to output the program exit status and run-time errors. We suggest redirecting both the standard output and standard error into files. With the bash shell (on Linux systems), you can do this using the command structure,

```
(spm [arguments] > out) >& err &
```

It may also be useful to redirect the standard input, especially is you're using `SPM` inside a batch job software, i.e.

```
(spm [arguments] > out < /dev/null) >& err &
```

On Microsoft Windows systems, you can redirect to standard output using,

```
spm [arguments] > out
```

And, on some Microsoft Windows systems (e.g., Windows7), you can redirect to both standard output and standard error, using the syntax,

```
spm [arguments] > out 2> err
```

Note that `SPM` outputs a few lines of header information to the output. The header consists of the program name and version, the arguments passed to `SPM` from the command line, the date and time that the program was called (derived from the system time), the user name, and the machine name (including the operating system and the process identification number). These can be used to track outputs as well as identifying the version of `SPM` used to run the model.

3.4. Command line arguments

The call to SPM is of the following form.:

spm [-c *config_file*] [*task*] [*options*]

-c *config_file* Define the input configuration file for SPM. If omitted, then SPM looks for a file named config.spm.

and where *task* is one of;

-h Display help (this page).

-l Display the reference for the software license (CPLv1.0).

-v Display the SPM version number.

-r *Run* the model once using the parameter values in the input configuration file, or optionally, with the values from the file denoted with the command line argument -i *file*.

-e Do a point *estimate* using the values in the input configuration file as the starting point for the parameters to be estimated, or optionally, with the start values from the file denoted with the command line argument -i *file*.

-p Do a likelihood *profile* using the parameter values in the input configuration file as the starting point, or optionally, with the start values from the file denoted with the command line argument -i *file*.

-m Do an *MCMC* estimate using the values in the input configuration file as the starting point for the parameters to be estimated, or optionally, with the start values from the file denoted with the command line argument -i *file*.

-s *number* *Simulate* the *number* of observation sets using values in the input configuration file as the parameter values, or optionally, with the values for the parameters denoted as estimated from the file with the command line argument -i *file*.

In addition, the following are optional arguments [*options*],

-i *file* *Input* one or more sets of free (estimated) parameter values from *file*. See Section 11.2.10 for details about the format of *file*.

-o *file* *Output* a report of the free (estimated) parameter values in a format suitable for -i *file*. See Section 11.2.10 for details about the format of *file*.

-t *number* Maximum number of *threads* to use when the model includes multi-threaded process.

-q Run *quietly*, i.e., suppress verbose printing of SPM.

-g *seed* Seed the random number *generator* with *seed*, a positive (long) integer value. Note, if -g is not specified, then SPM will generate a random number seed based on the computer clock time.

3.5. Constructing an SPM input configuration file

The model definition, parameters, observations, and reports are specified in an input configuration file. The population section is described in Section 4 and the population commands in Section 8. Similarly, the estimation section is described in Section 5 and its commands in Section 9, and in Section 7 and Section 11 for the report and report commands.

3.5.1. Commands

SPM has a range of commands that define the model structure, processes, observations, and how tasks are carried out. There are three types of commands,

1. Commands that have an argument and do not have subcommands (for example, `@include file`)
2. Commands that have a label and subcommands (for example `@process` must have a label, and has subcommands)
3. Commands that do not have either a label or argument, but have subcommands (for example `@model`)

Commands that have a label must have a unique label, i.e., the label cannot be used on more than one command of that type. The labels must start with a letter or underscore, can contain letters, underscores, or numbers. Labels must not contain white-space, a full-point ('.'), or other characters that are not letters, numbers, or an underscore.

3.5.2. Subcommands

Subcommands in SPM are for defining options and parameter values for commands. They always take an argument which is one of a specific *type*. The types acceptable for each subcommand are defined in Section 8.12.11, and are summarised below.

Like commands (`@command`), subcommands and their arguments are not order specific — except that that all subcommands of a given command must appear before the next `@command` block. SPM may report an error if they are not supplied in this way, however, in some circumstances a different order may result in a valid, but unintended set of actions, leading to possible errors in your expected results.

The arguments for a subcommand are either,

switch true/false

integer an integer number

integer vector a vector of integer numbers

integer range a range of integer numbers separated by a hyphen (-), for example 1994-1996 2000 is expanded to an integer vector of values 1994 1995 1996 2000).

constant a real number (i.e., double)

constant vector a vector of real numbers (i.e., vector of doubles)

estimable a real number that can be estimated (i.e., estimable double)

estimable vector a vector of real numbers that can be estimated (i.e., vector of estimable doubles)

string a categorical (string) value

string vector a vector of categorical values

Switches are parameters which are either true or false. Enter *true* as `true` or `t`, and *false* as `false` or `f`.

Integers must be entered as integers (i.e., if `year` is an integer then use 2008, not 2008.0)

Arguments of type integer vector, integer range, constant vector, estimable vector, or categorical vector contain one or more entries on a row, separated by white space (tabs or spaces).

Estimable parameters are those parameters that SPM can estimate, if requested. If a particular parameter is not being estimated in a particular model run, then it acts as a constant. Within SPM only estimable parameters can be estimated. And, you have to tell SPM those that are to be estimated in any particular model. Estimable parameters that are being estimated within a particular model run are called the *estimated parameters*.

3.5.3. The command-block format

Each command-block either consists of a single command (starting with the symbol) and, for most commands, a label or an argument. Each command is then followed by its subcommands and their arguments, e.g.,

`@command`, or

`@command argument`, or

`@command label`

and then

`subcommand argument`

`subcommand argument`

etc,.

Blank lines are ignored, as is extra white space (i.e., tabs and spaces) between arguments. But don't put extra white space before a `@` character (which must also be the first character on the line), and make sure the file ends with a carriage return.

There is no need to mark the end of a command block. This is automatically recognized by either the end of the file, section, or the start of the next command block (which is marked by the `@` on the first character of a line). Note, however, that the `@include` is the only exception to this rule. See Section 12) for details of the use of `@include`.

Note that in the input configuration file, commands, sub-commands, and arguments are not case sensitive. However, labels and variable values are case sensitive. Also note that if you are on a Linux system then external calls to files are case sensitive (i.e., when using `@include file`, the argument `file` will be case sensitive).

Characters used in labels must be alphanumeric and can include underscores (_). Other characters will result in an error.

3.5.4. Commenting out lines

Text that follows a # on a line are considered to be comments and are ignored. If you want to remove a group of commands or subcommands using #, then comment out all lines in the block, not just the first line.

Alternatively, you can comment out an entire block or section by placing curly brackets around the text that you want to comment out. Put in a { as the first character on the line to start the comment block, then end it with }. All lines (including line breaks) between { and } inclusive are ignored.

(These should ideally be the first character on a line. But if not, then the entire line will be treated as part of the comment block.)

3.5.5. Determining parameter names

When SPM processes a input configuration file, it translates each command and each subcommand into a parameter with a unique name. For commands, this parameter name is simply the command name. For subcommands, the parameter name format is either,

`command[label].subcommand` if the command has a label, or

`command.subcommand` if the command has no label, or

`command[label].subcommand(i)` if the command has a label and the subcommand arguments are a vector, and we are accessing the *i*th element of that vector.

The unique parameter name is used to reference the parameter when estimating, applying a penalty, or applying a profile. For example, the parameter name of subcommand `r0` of the command `@process` with the label `MyRecruitment` is,

```
process[MyRecruitment].r0
```

3.6. SPM exit status values

When SPM completes its task successfully or errors out gracefully, it returns a single exit status value (0) to the operating system. The operating system will return (-1) if SPM terminates unexpectedly. To determine if SPM has completed its task successfully, check the standard output for error and information messages.

4. The population section

4.1. Introduction

The population section specifies the model structure, movement and population dynamics, and other associated parameters. It describes the model structure (both the spatial and population structure), defines the population (for example, recruitment, migration, and mortality) and movement processes, defines the layers (the known attributes of each spatial cell), selectivities, and model parameters.

The population section consists of several components, including;

- The spatial and population structure
- Model initialisation (i.e., the state of the model at the start of the first year)
- The years over which the model runs (i.e., the start and end years of the model)
- The annual cycle (time-steps and processes that are applied in each time-step)
- The specifications and parameters of the processes;
  - Population processes (i.e., processes that add, remove individuals to or from the partition, or shift numbers between ages and categories in the partition)
  - Spatial processes (i.e., processes that move or shift cohorts between spatial locations but do not alter their ages or categories)
- Layers (used by processes, observations and reports) and their definitions
- Selectivities
- Parameter values and their definitions
- Derived quantities, quantities by cell and meta-layers required as parameters for some processes (i.e., spawning stock biomass to resolve the spawner-recruit relationship in a recruitment process)

4.2. Spatial structure

The spatial structure of `SPM` is represented by an $n_{rows} \times n_{cols}$ grid, with rows $i = 1 \ldots n_{rows}$ and columns $j = 1 \ldots n_{cols}$. Each cell of this matrix records the population structure at that point in space, where the population structure is represented by an $n_{categories} \times n_{ages}$ rectangular matrix (with categories $k = 1 \ldots n_{categories}$ and ages $l = 1 \ldots n_{ages} = age_{min} \ldots age_{max}$. Hence we can describe any spatial and population element of the model as element(i, j, k, l). We define, within the spatial grid ($n_{rows} \times n_{cols}$), locations where the population can and cannot potentially be present using a *layer*.

`SPM` implements a single spatial structure, a grid of *square* cells (Figure 4.1). The spatial grid can be of an arbitrary size, but must be rectangular.

The dimensions of the spatial grid are user defined but must be at least a 1×1 grid (i.e., a single spatial cell), and the largest spatial structure currently allowed by `SPM` is a grid of 1000×1000 cells – although we note that models of this size are untested and will probably have very long run times.

Associated with the $n_{rows} \times n_{cols}$ spatial structure is the one compulsory layer (see Section 4.4), the *base layer*. This defines the locations where the population can and cannot potentially be present (e.g., in a marine model, the locations associated with the sea and not land). These are defined as the cells where the base layer has a value greater than zero. There must be at least one cell in the

spatial grid where the population can be present. In addition, the base layer also defines the relative *area* of each spatial cell that is used for density calculations within SPM.

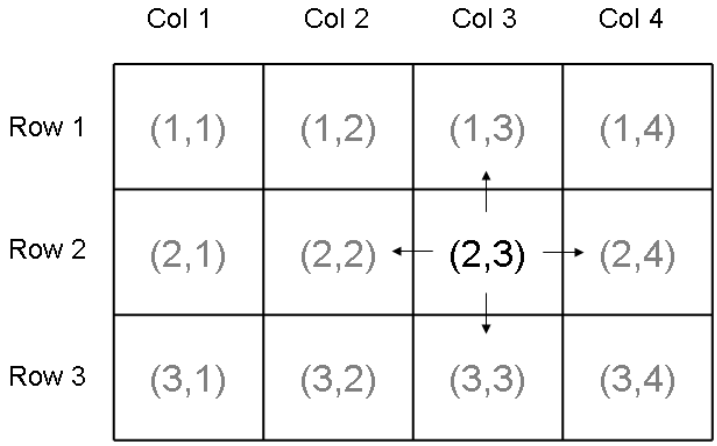


Figure 4.1: An illustration of the spatial structure

Models are implemented as a grid of square cells making up a rectangular matrix. Distance between cells is determined as the euclidean distance between cell centres, modified by an arbitrary scalar.

Hence, the definition of the spatial structure includes;

- The type of spatial grid and its dimensions, n_{rows} and n_{cols}
- The label of a numeric layer to be used as the base layer (defining the locations where the population can be present as well as the area of each cell)
- The length (distance) of a side of the grid cell to be used as the scaler for distance calculations

For example, to specify a model with 3 rows and 4 columns (i.e., 12 spatial cells) with a base layer called `base` and 2 km sides of each cell, then the syntax for `@model` would include,

```
@model
nrows 3
ncols 4
layer base
cell_length 2
```

See below for how to define a layer using `@layer`.

4.3. Population structure

The population structure in SPM is represented by a matrix containing an arbitrary number of user defined categories (rows), and an arbitrary age range (columns). Hence, each spatial cell has a population state described as $n_{categories} \times n_{ages}$ rectangular matrix with categories $k = 1 \ldots n_{categories}$ and ages $l = age_{min} \ldots age_{max}$.

The names and number of categories are user defined, but there must be at least one category defined for a model. The ages are defined as a sequence from age_{min} to age_{max}, with the last age optionally a plus group. In order to calculate biomass, the age-size relationship for each category must also be defined.

Hence, the definition of the population structure includes;

- The number and labels of the categories, $k_{categories}$
- The age_size relationship for each category
- The minimum and maximum ages that define the ages of the model, l_{ages}
- If the last age is a plus group

For example, to specify a model with 2 categories (male and female) with ages 1-20 (with the last age a plus group) and an age-size relationship defined with the label `male-growth` and `female-growth`, then the `@model` example from above becomes,

```
@model
nrows 3
ncols 4
layer base
cell_length 2
categories male female
min_age 1
max_age 20
age_plus_group True
age_size male-growth female-growth
```

See below for how to define age-size relationships using `@age_size`.

4.4. Layers

Layers are a key underlying concept in `SPM`. They comprise of a grid of known values, with a value for every spatial cell in the model. Layers are used by processes, observations, and outputs commands to supply spatially explicit covariates and any categorical groupings required.

Layers are used by `SPM` to evaluate locations where the population may be present (via the *base layer*), to provide sets of known attributes or values of each spatial location (for some processes and for preference based movements), and to group or categorise cells for use by processes and observations. Layers consist of an $n_{rows} \times n_{cols}$ matrix and can be either *numeric* or *categorical*.

Every model must define at least one layer, the base layer L_B. A layer is defined as a $n_{rows} \times n_{cols}$ matrix of values (albeit with the exception of layers that describe distance — these are described in detail below), where the value in each cell represents a known quantity. For example layers may represent classifications, physical attributes, or some other known or assumed quantity. Typically they are provided by the user as a matrix of values, although some layers (e.g., abundance or distance layers) can be calculated by `SPM` during a model run.

Within `SPM`, layers are used in the following contexts:

1. The base layer: The base layer L_B is a special layer (there must be exactly one base layer defined within the model) that defines the locations where the population can and cannot potentially be present (e.g., locations associated with the sea and not land in a marine model). Here, we define that a cell may potentially have part of the population present if every element $L_B(i, j) \geq 0$. Further, positive values of the base layer L_B represent the *area* represented by that spatial cell. Note, the values in the base layer must be numeric, but cannot be a meta-layer (*see below*).
2. Covariate layers: A model may have many covariate layers, and these are used as covariates of some population or movement process (e.g., the sea floor depth may be a covariate of

some movement process). The values in layers used as covariates can be either numeric or categorical.

3. Classification layers: A model may have many classification layers, and these are used as a classification or grouping variable for aggregating data over individual spatial cells (i, j), e.g., statistical areas or management areas. Such layers are typically used to aggregate the population within cells into groups so-as to allow comparison with observations. The values in layers used as classification layers must be categorical.

`SPM` defines the following types of layer;

1. Numeric layers: A model may have many numeric layers, and these can be used as covariates of a population or movement process (e.g., depth may be a covariate of some movement process), and/or locations of event mortality. Numeric layers can contain only continuous (numeric) variables. Values for a numeric layer must be supplied for each cell by the user. Numeric layers can be rescaled to sum to some user-defined value, and unlike other layers, the values for each cell can be estimated.

   For example, to specify a numeric layer for a spatial model with 3×4 cells called, say `base`, with the top left and bottom right cells set to zero and all other cells set to one and not rescaled, use,

   ```
   @layer base
   type numeric
   data 0 1 1 1
   data 1 1 1 1
   data 1 1 1 0
   ```

2. Categorical layers: A model may have many categorical layers, and these can be used as a classification or grouping variable for aggregating data over individual cells, e.g., management areas; or as covariates of a population or movement process. Such layers are typically used to aggregate the population within cells into groups for comparing with observations, or to apply specific movement characteristics. The values in layers used as categorical layers can contain any characters (except white space), and are interpreted as categorical values. Values for a categorical layer must be supplied for each cell by the user.

   For example, to specify a categorical layer for a spatial model with 3×4 cells called, say `zone`, with the top left cells allocated as zone `A`, the bottom right allocated as zone `C`, and the rest as zone `B`, then use,

   ```
   @layer zone
   type categorical
   data A A B B
   data A A C C
   data B B C C
   ```

3. Distance layers: A distance layer is one that defines the distance between cells. `SPM` has four types of distance layer: (i) the Euclidean distance between cell centres (`distance`); (ii) the distance between two locations (defined by longitude and latitude) using the haversine formula (`haversine`); (iii) the shortest path between cells by traversing cell centres using Dijkstra's algorithm (Dijkstra, 1959) and assumes that cells excluded by the base layer are not traversable (`dijkstra`); and (iv) the shortest path between cells using the Dijkstra's algorithm but with the distances calculated using the haversine formula from latitudes and longitudes (`haversine_dijkstra`).

   The Euclidean distance method simply calculates the values of the distance layer as the Euclidean distance between the centres of every pair of cells, taking into account the length

of the side of a cell. Here, the distance between cell a and cell b can be defined as,

$$d(a,b) = \lambda\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \tag{4.1}$$

where x and y represent the x- and y-coordinates of a and b respectively, and λ is an arbitrary scaler representing the length of one side of the square cell. Unlike other types of layers, distance layers are not a $n_{rows} \times n_{cols}$ grid of values, but rather a matrix of dimension $(n_{rows} \times n_{cols}) \times (n_{rows} \times n_{cols})$ where the distance between each cell and every other cell is evaluated. Note that the distance between any cell and itself is 0.

For example, to specify a Euclidean distance layer, use,

```
@layer Distance
type distance
```

The haversine method calculates the great-circle distance between two points on a sphere from the longitudes and latitudes of the cells, with the assumption that the Earth is a perfect sphere with radius exactly 6371 km. Two numeric layers are required, giving the longitude and latitude of each cell respectively. Latitude values must be between -90° (southern latitudes) and 90° (northern latitudes). Longitude values must be between 0° and 360°. Note that the distance between any cell and itself is 0.

For example, to specify a haversine distance layer for a 2×3 spatial model, assuming that latitudes and longitudes of each cell are given in the numeric layers labelled `cell_latitudes` and `cell_longitudes`, then use,

```
@layer cell_latitudes
type numeric
data -45 -45 -45
data -46 -46 -46

@layer cell_longitudes
type numeric
data 176 177 178
data 176 177 178

@layer Haversine_distance
type haversine
latitude cell_latitudes
longitude cell_longitudes
```

The Dijkstra method calculates the distance between two points as the sum of the shortest path between cell centres that does not traverse through cells excluded by the base layer. Distances between adjacent cells are calculated using Euclidean distance as described above with λ an arbitrary scaler representing the length of one side of each cell. Note that the distance between any cell and itself is 0.

For example, to specify a Dijkstra distance layer then use,

```
@layer Dijkstra_distance
type dijkstra
```

The haversine-Dijkstra method calculates the distance between two points as the sum of the shortest path between cell centres that does not traverse through cells excluded by the base layer, where distances are calculated using the haversine formulae. Distances between adjacent cells are calculated using haversine distance as the great-circle distance between two points on a sphere from the longitudes and latitudes of the cell, with the assumption that the Earth is a perfect sphere with radius exactly 6371 km. Two numeric layers are required, giving

the longitude and latitude of each cell respectively. Latitude values must be between -90° (southern latitudes) and 90° (northern latitudes). Longitude values must be between 0° and 360°. Note that the distance between any cell and itself is 0.

For example, to specify a haversine Dijkstra distance layer for a 2×3 spatial model, assuming that latitudes and longitudes of each cell are given in the numeric layers labelled `cell_latitudes` and `cell_longitudes`, then use,

```
@layer cell_latitudes
type numeric
data -45 -45 -45
data -46 -46 -46

@layer cell_longitudes
type numeric
data 176 177 178
data 176 177 178

@layer Haversine_Dijkstra
type haversine_dijkstra
latitude cell_latitudes
longitude cell_longitudes
```

4. Abundance layers: The abundance layer is the sum of the number of individuals within cell a in categories k and with selectivity S_l at age l.

$$N(a) = \sum_k \sum_l S_l \text{ element}(i,j,k,l) \tag{4.2}$$

`SPM` calculates the values of the layer when running the model at the point in time where the value is required.

For example, to specify an abundance layer of all individuals who are categorised as `mature`, use,

```
@layer Abundance
type abundance
categories mature
selectivities One
```

5. Biomass layers: The biomass layer is the sum of the biomass of individuals within cell a in categories k, with selectivity S_l at age l, and mean weight w_{kl}

$$N(a) = \sum_k \sum_l w_{k,l} S_l \text{ element}(i,j,k,l) \tag{4.3}$$

`SPM` calculates the values of the layer when running the model at the point in time where the value is required.

For example, to specify a biomass layer of all individuals who are categorised as `mature`, use,

```
@layer Biomass
type biomass
categories mature
selectivities One
```

6. Abundance-density layers: The abundance density layer is the density of the number of individuals within cell a with area A_a in categories k, with selectivity S_l at age l,

$$N(a) = \frac{1}{A_a} \sum_k \sum_l S_l \text{ element}(i,j,k,l) \tag{4.4}$$

SPM calculates the values of the layer when running the model at the point in time where the value is required.

For example, to specify an abundance density layer of all individuals who are categorised as mature, use,

```
@layer AbundanceDensity
type abundance_density
categories mature
selectivities One
```

7. Biomass-density layers: The biomass-density layer is the density of the biomass of individuals within cell a with area A_a in categories k, with selectivity S_l at age l, and mean weight w_{kl},

$$N(a) = \frac{1}{A_a} \sum_k \sum_l w_{k,l} S_l \text{ element}(i, j, k, l) \tag{4.5}$$

SPM calculates the values of the layer when running the model at the point in time where the value is required.

For example, to specify a biomass density layer of all individuals who are categorised as mature, use,

```
@layer BiomassDensity
type bioomass_density
categories mature
selectivities One
```

8. Derived quantity and derived quantity by cell layers: SPM can use values from a derived quantity or a derived quantity by cell as a layer. These are the values of a derived quantity, and require an offset (in years) to extract the desired value for each year. They will extract the derived quantity either for each cell (for derived quantities by cell) or as a total value applied to each cell (for ordinary derived quantities) as values for a layer.

For example, to specify a layer of the biomass of all individuals who are categorised as mature in time step StepOne from two years ago, first define a derived quantity,

```
@derived_quantity_by_cell Mature
type biomass
categories mature
selectivities One
time_step StepOne
```

Then define the derived layer,

```
@layer MatureBiomass
type derived_quantity
derived_quantity Mature
year_offset 2
```

9. Meta-layers: SPM defines a special type of layer known as a *meta-layer*. Meta-layers allows individual layers to be indexed by year and applied as an annually varying layer within the model. For example, assume a model that uses Sea Surface Temperature (SST) as a layer, perhaps to drive some movement process. The SST values for each year of the model would be defined as individual layers, each with a unique label. A meta-layer could be defined that indexed the individual annual SST layers by year, and used as a covariate layer in the movement process. Meta-layers have a *default* layer that is used for time periods that are not specifically defined. Meta layers can be used wherever ordinary layers are used (except that they cannot be used as the base layer), with SPM extracting the appropriate layer value corresponding to the year or the initialisation phase.

(a) Numeric meta-layers: Numeric meta-layers are a meta layer of numeric layers — the individual ordinary layers that make up the meta-layer must all be of numeric type. For example, assuming that the layers `SST` and `SST1990`, `SST1995`, etc., are defined elsewhere, then to specify a numeric meta-layer with specific values for the years 1990-1995, and a default for all other years (including the initialisation phases), use,

```
@layer AnnualNumericLayer
type numeric_meta
default_layer SST
years    1990    1991    1992    1993    1994    1995
layers SST1990 SST1991 SST1992 SST1993 SST1994 SST1995
```

(b) Categorical meta-layers: Categorical meta-layers are a meta layer of categorical layers — the individual ordinary layers that make up the meta-layer must all be of categorical type. Categorical meta-layers are specified in the same way as numeric meta-layers. For example, assuming that the layers `zones` and `zone1990`, `zone1995`, etc., are defined elsewhere, then to specify a categorical meta-layer with specific values for the years 1990-1995, and a default for all other years (including the initialisation phases), use,

```
@layer AnnualCategoricalLayer
type categorical_meta
default_layer zones
years     1990     1991     1992     1993     1994     1995
layers zone1990 zone1991 zone1992 zone1993 zone1994 zone1995
```

4.5. Time sequences

The time sequence of the model is defined in two parts;

- Initialisation
- Run years

4.5.1. Annual cycle

The annual cycle is implemented as a set of processes that occur, in a user-defined order, within each year. time-steps are used to break the annual cycle into separate components, and allow observations to be associated with different sets of processes. Any number of processes can occur within each time-step, in any order and can occur multiple times within each time-step. Note that time-steps during the initialisation phases can be different from that which is applied during the model years.

4.5.2. Initialisation

Model initialisation can occur in several phases, each which iterates through a number of years carrying out the population and/or spatial processes defined for that phase. At the end of the initialisation step, `SPM` runs through the model years carrying out processes in the order defined in the annual cycle, and can evaluate expected values of observations in order to calculate likelihoods, or simulate observations from the current state.

`SPM` initialises the initial equilibrium state as an iterative process: a general solution that initialises complex structured population and movement models can be difficult to implement using analytic techniques. However, initialising via iteration for a long-lived species with complex movements can take many iterations and hence be slow to run. In `SPM`, we allow for user-defined multi-phased

initialisation using iteration to allow the user to optimize models for speed. Each phase of the initialisation can involve any number of population and/or movement processes. Note that the length of the initialisation period may affect the model outputs, and that a period should be chosen to allow the population state to converge.

In addition, each initialisation process can optionally be stopped early if a user defined convergence criteria is met. For a set of user defined years in the initialisation phase, convergence is defined as met if the proportional absolute summed difference between the state in year $t-1$ and the state in year t ($\widehat{\lambda}$) is less than λ where,

$$\widehat{\lambda} = \frac{\sum_i \sum_j \sum_k \sum_l |\text{element}(i,j,k,l)_t - \text{element}(i,j,k,l)_{t-1}|}{\sum_i \sum_j \sum_k \sum_l \text{element}(i,j,k,l)_t} \tag{4.6}$$

Note that the check itself can be expensive time-wise, hence options exist for the user to request if or when this convergence check is carried out.

In each initialisation phase, the processes defined for that phase are carried out and used as the starting point for the following phase or, if it is the last phase, then the years that the model is run over. The first phase is always initialised with each element (i.e., each age and category within each spatial cell) set at zero. Note that this means that recruitment processes where the numbers of recruits is based on a stock recruitment or density dependant relationship will likely fail if used in the first phase of an initialisation.

The multi-phase iteration also allows the user to determine if the initialisation has converged in a particular model run. Here, add an additional initialisation phase for, say, 1 year as the last initialisation phase (with the same processes applied). Then, using the initialisation reports (`@report[label].type=initialisation_phase`), print a copy of the partition just before and just after that phase. If the initialisation has converged to an equilibrium state, then the partition at both these time intervals will be the same.

Hence, for initialisation you need to define;

- The initialisation phases
- The number of years in each phase and the processes to apply in each

For example, to specify a model with a single phase of initialisation for 100 years (with a conve4rgence check at 80, 90, and 100 years) using time-steps `one` and `two`, then the `@model` example from above becomes,

```
@model
nrows 3
ncols 4
layer base
cell_length 2
categories male female
min_age 1
max_age 20
age_plus_group True
age_size male-growth female-growth
initialisation_phases phase1
```

with the initialisation phases specified as,

```
@initialisation_phase phase1
```

```
years 100
time_steps one two
lambda 1e-10
lambda_years 80 90 100
```

4.5.3. Model years

Following initialisation, the model then runs over a number of user-defined years. For this part of the model, the annual cycle can be broken into separate time-steps (see below). The model runs from the start of year `initial` and runs to the end of year `current`.

For example, to specify a model for the years 1990–2010 , then the `@model` example from above becomes,

```
@model
nrows 3
ncols 4
layer base
cell_length 2
categories male female
min_age 1
max_age 20
age_plus_group True
age_size male-growth female-growth
initialisation_phases phase1
initial_year 1990
current_year 2010
```

4.5.4. Time-steps

The annual cycle is run during initialisation and over the model years. It is broken into time-steps. Within each time-step, processes are carried out in the order specified, and can be the same or different to processes in other time-steps or in the initialisation phases of the model. Further, observations (see later) can be associated with the state of the model at the end of a time-step — i.e., time-steps are used to break the annual cycle into discrete 'chunks', and to allow the model to appropriately generate expected values for observations.

For example, to specify a model with the time-steps `one` and `two`, and the processes `myRecruitment` and `myMaturation` in `one` and processes `myAgeing` and `myMortality` on `two`, then the `@model` example from above becomes,

```
@model
nrows 3
ncols 4
layer base
cell_length 2
categories male female
min_age 1
max_age 20
age_plus_group True
age_size male-growth female-growth
initialisation_phases phase1
initial_year 1990
current_year 2010
time\_steps one two
```

with the time-step definitions

```
@time_step one
processes myRecruitment myMaturation

@time_step two
processes myAgeing myMortality
```

See later for how to define processes using `@process`.

4.6. Processes

Processes produce changes in the model partition, by adding, removing or moving individuals between spatial cells (movement processes), and ages or categories (population processes). These include processes such as recruitment, mortality, ageing, and various forms of movement.

`SPM` has two types of processes, *population* and *movement* processes. Population processes are those processes which modify, move or otherwise change the numbers of individuals *within* a spatial cell, i.e., they do not affect the spatial location of a cohort. Movement processes, on the other hand, move, shift or otherwise modify cohorts *between* spatial cells, but do not affect the age or category of the numbers in each cohort.

The population processes include recruitment, ageing, mortality events (e.g., natural and exploitation) and category transition processes (i.e., processes that move individuals between categories, while preserving their age structure). See Section 4 for a complete list of available processes.

Each of these processes is carried out in the user-defined prescribed order when initialising the model, and then for a user-defined order in each year in the annual cycle.

`SPM` implements three different types of movement processes;

1. A migration movement rate of cohorts between any two locations, and is roughly analogous to movements between areas as implemented in other population models, such as CASAL (Bull et al., 2012).
2. An adjacent cell movements, parametrised by some function of an underlying layer — equivalent to, for example, movement processes implemented in Fish Heaven (Ball and Constable, 2000, Ball and Williamson, 2003).
3. Movement parametrised as a probability density function. Here, the key underlying idea is that the spatial distribution of cohorts at any point in time and at any location can be represented as a density function based on attributes of that location, local abundance, and/or distance from their previous location (Bentley et al., 2004a,b).

An `SPM` model can be parametrised by both population processes (for example, ageing, recruitment, and mortality), and movement processes. Population processes are those processes which modify, move or otherwise change the numbers of individuals within a spatial cell, i.e., they do not affect the spatial location of a cohort. Movement processes, on the other hand, move, shift or otherwise modify cohorts between spatial cells, but do not affect the age or category of the numbers in each cohort.

4.7. Population processes

Population processes are those processes that change the population state of individuals, but retain their location. The population processes are described below.

4.7.1. Recruitment

Recruitment processes are defined as a process that introduces new individuals into the model. `SPM` implements four types of recruitment process: constant recruitment; Beverton-Holt recruitment (Beverton and Holt, 1957); local Beverton-Holt recruitment; and recruitment proportional to an abundance or biomass .

In the recruitment processes, the number of individuals are added to a single age class within the partition, with the amount defined by the type of recruitment process and its function. If more than one category is defined, then the proportion of recruiting individuals to be added to each category is specified by the `proportions` parameter. For example, if recruiting to categories labelled male and female, then you might set the proportions as 0.5 and 0.5 respectively to denote that half of the recruits recruit to the male category and the remaining half to the female category.

Recruitment occurs in those cells defined by a layer, or all cells if a layer has not been specified.

For the constant, Beverton-Holt, and proportional recruitment processes, the number of individuals in cell cell(i,j) following recruitment in year y is,

$$\text{element}(i,j,k,l) \leftarrow \text{element}(i,j,k,l) + p_k(R_y/n)\frac{L_i j}{\sum_i j L_i j} \tag{4.7}$$

where age is the age defined as the recruitment age, p_k is the proportion to category k defined to have recruitment, n is the number of spatial locations where recruitment occurs, and the recruitment to each cell is scaled to be proportional to the value of the layer in that cell. See below for how R_y is determined in each of these cases.

In the local Beverton-Holt recruitment process, individuals are recruited to an individual cell, based on the local abundance or biomass (i.e., the local SSB). For each cell where cell(i,j) is a member of some layer L_R, the number of individuals in that cell following recruitment in each year y is

$$\text{element}(i,j,k,l) \leftarrow \text{element}(i,j,k,l) + p_k R_y L_{ij} \tag{4.8}$$

where age is the age defined as the recruitment age, p_k is the proportion recruitment to category k defined to have recruitment, and the recruitment to each cell is the product of the Beverton-Holt stock recruitment relationship (R_y).

Constant Recruitment

In the constant recruitment process the total number of recruits added each year is R_y, and is simply R_0, i.e.,

$$R_y = R_0 \tag{4.9}$$

It is equivalent to a Beverton-Holt recruitment process where steepness is set equal to one ($h = 1$).

For example, to specify a constant recruitment process, where individuals are added to the category 'immature' at $age = 1$, and the number to add is $R_0 = 5 \times 10^5$ in areas proportional to the value of the layer `recruitment`, then the syntax is,

```
@process Recruitment
type constant_recruitment
categories immature
proportions 1.0
R0 500000
age 1
layer recruitment
```

Beverton-Holt recruitment

In the Beverton-Holt recruitment process the total number of recruits added each year is R_y, and is the product of the average recruitment R_0, the annual year class strength multiplier, YCS, and the stock-recruit relationship i.e.,

$$R_y = R_0 \times YCS_{y-offset} \times SR(SSB_{y-offset}) \tag{4.10}$$

where $offset$ is the number of years offset to link the year class with the year of spawning y, and SR is the Beverton-Holt stock-recruit relationship parametrised by the steepness h,

$$SR(SSB_y) = \frac{SSB_y}{B_0} / \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB_y}{B_0}\right)\right) \tag{4.11}$$

Note that the Beverton-Holt recruitment process requires a value for B_0 and SSB_y to resolve the stock-recruitment relationship. Here, a derived quantity (see Section 4.9) must be defined that provides the annual SSB_y for the recruitment process. B_0 is then defined as the value of the SSB at the end of one of the initialisation phases. During initialisation the YSC multipliers are assumed to be equal to one, and recruitment that happens in the initialisation phases that occur before and during the phase when B_0 is determined is assumed to have steepness $h = 1$ (i.e. in those initialisation phases, recruitment is simply equal to R_0). Note it is an error if $B_0 <= 0$.

Recruitment in the initialisation phases after the phase where B_0 follows the Beverton-Holt stock-recruit relationship defined above. Recruits are then distributed across cells in proportion to the values in a numeric layer.

Year classes are standardised to be equal to one over the period S defined by `standardise_YCS_years`, i.e., the year classes (YCS) for each year of the model are calculated as

$$YCS_i = \begin{cases} Y_i / mean_{y \in S}(Y_y) & : y \in S \\ Y_i & : y \notin S \end{cases} \tag{4.12}$$

Note that the an effect of this parameterisation is that R_0 is then defined as the mean estimated recruitment over the years S, because the mean year class multiplier over these years will always be one.

For example, assume a Beverton-Holt recruitment process, where individuals are added to the category 'immature' at $age = 1$, the number to add is $R_0 = 5 \times 10^5$ in areas proportional to the value of the layer `MyRecruitment`. Then `SSB_Biomass` is a derived quantity that specifies the total spawning stock biomass, with B_0 the value of the derived quantity at the end of the initialisation phase labelled `phase1`. The YCS are standardised to have mean one in the period 1994 to 2004, and recruits enter into the model two years following spawning. Then the command specification is,

```
@process Recruitment
type BH_recruitment
categories immature
```

```
proportions 1.0
R0 500000
steepness 0.75
age 1
layer MyRecruitment
B0 phase1
SSB SSB_Biomass
standardise_YCS_years 1994-2004
YCS_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
YCS_values   1    1    1    1    1    1    1    1    1    1    1    1    1
SSB_offset 2
```

Note that if not specified, SSB_offset is set at the value of age. This corresponds to cases where recruitment happens after spawning. SSB_offset can be user-defined to a different value if needed — for example, if recruitment happens in a time-step before spawning, the offset will need to be specified as $age+1$.

Local Beverton-Holt recruitment

The local Beverton-Holt recruitment process assumes that, for recruitment, each cell acts like a local population and is independent of its neighbours. The value of the SSB^i_y in each cell i for each year y is used to determine the amount of recruitment that enters that cell. If desired, these local recruits can subsequently be subjected to movement (see Section 4.8).

If the recruitment to cell i is R^i_y, then R^i_y is the product of the average recruitment R^i_0 for that cell, the annual year class strength multiplier (assumed to be the same for all cells in each year), and the stock-recruit relationship, i.e.,

$$R^i_y = R^i_0 \times YCS_{y-offset} \times SR(SSB^i_{y-offset}) \tag{4.13}$$

where $offset$ is the number of years offset to link the year class with the year of spawning, and SR is the Beverton-Holt stock-recruit relationship parametrised by the steepness h and initial biomass B^i_0

$$SR(SSB^i) = \frac{SSB^i}{B^i_0} \Big/ \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB^i}{B^i_0}\right)\right) \tag{4.14}$$

Note that the local Beverton-Holt recruitment process requires a value for B^i_0 and SSB^i_y to resolve the stock-recruitment relationship. Here, a derived quantity by cell (see Section 4.10) must be provided that defines B_0 and the annual SSB_y for the recruitment process. As for the Beverton-Holt recruitment process above, B_0 is then defined as the value of the SSB at the end of one of the initialisation phases. During initialisation the YSC multipliers are assumed to be equal to one, and recruitment that happens in the initialisation phases that occur before and during the phase when B_0 is determined is assumed to have steepness $h = 1$ (i.e., in those initialisation phases, recruitment is simply equal to R_0). Recruitment in the initialisation phases after the phase where B_0 was determined follow the Beverton-Holt stock-recruit relationship defined above. If a layer is defined for recruitment, R^i_0 is defined for each cell as product of R_0 and the layer value in each cell.

Year classes are standardised to be equal to one over the period S defined by `standardise_YCS_years`, i.e., the year classes (YCS) for each year of the model are calculated as

$$YCS_i = \begin{cases} Y_i / mean_{y \in S}(Y_y) & : y \in S \\ Y_i & : y \notin S \end{cases} \tag{4.15}$$

Note that the an effect of this parameterisation is that R_0^i is then defined as the mean estimated recruitment over the years S, because the mean year class multiplier over these years will always be one.

For example, assume a local Beverton-Holt recruitment process, where individuals are added to the category 'immature' at $age = 1$, the number to recruit in each cell is $R_0 = 5 \times 10^5$ multiplied by the proportional value of the layer `MyRecruitment`. Then `SSB_Biomass` is a derived quantity by cell that specifies the spawning stock biomass in each cell, with B_0 the value of the derived quantity at the end of the initialisation phase labelled `phase1`. The YCS are standardised to have mean one in the period 1994 to 2004, and recruits enter into the model two years following spawning. Then the command specification is,

```
@process Recruitment
type local_BH_recruitment
categories immature
proportions 1.0
R0 500000
steepness 0.75
age 1
layer MyRecruitment
B0 phase1
SSB SSB_Biomass
standardise_YCS_years 1994-2004
YCS_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
YCS_values   1    1    1    1    1    1    1    1    1    1    1    1    1
SSB_offset 2
```

Note that if not specified, SSB_offset is set at the value of age. This corresponds to cases where recruitment happens after spawning. SSB_offset can be user-defined to a different value if needed — for example, if recruitment happens in a time-step before spawning, the offset will need to be specified as $age + 1$.

Proportional recruitment

In the proportional recruitment process the total number of recruits added each year is R_y, and is the product of a multiplier (λ) and a measure of abundance or biomass, i.e.,

$$R_y = \lambda \times SSB_{y-offset} \tag{4.16}$$

where $offset$ is the number of years offset to link the year class with the year of spawning y.

Note that the proportional recruitment process requires a value for SSB_y in each year, with a default value that is applied in the initialisation phases where SSB_y is not be available. Here, a derived quantity (see Section 4.9) must be defined that provides the annual SSB_y for the recruitment process. B_0 is then defined as the value of the SSB at the end of one of the initialisation phases. During initialisation the recruitment that happens in the initialisation phases that occur before and during the phase when B_0 is determined is assumed to be R_0. Recruitment in the initialisation phases after the phase where B_0 was determined follow the relationship defined above. Recruits are then distributed across cells in proportion to the values in a numeric layer.

For example, assume a proportional recruitment process, where individuals are added to the category 'immature' at $age = 1$, the number to add is 0.5 the female breeding abundance, and placed into areas in proportion to the value of the layer `MyRecruitment`. The `female_abundance` is a derived quantity by cell that specifies the total number of breeding females, with B_0 the value of the derived quantity at the end of the initialisation phase labelled `phase1`. Recruits enter into the model two years following spawning. Then the command specification is,

```
@process Recruitment
type proportional_recruitment
categories immature
proportions 1.0
lambda 0.5
R0 500000
age 1
layer MyRecruitment
B0 phase1
SSB female_abundance
SSB_offset 2
```

Note that if not specified, SSB_offset is set at the value of age. This corresponds to cases where recruitment happens after spawning. SSB_offset can be user-defined to a different value if needed — for example, if recruitment happens in a time-step before spawning, the offset will need to be specified as $age+1$.

4.7.2. Ageing

The ageing process simply moves all individuals in the named categories to the next age class. The ageing process is defined as,

$$\text{element}(i,j,k,l) \leftarrow \text{element}(i,j,k,l-1) \tag{4.17}$$

except that in the case of the plus group (if defined),

$$\text{element}(i,j,k,age_{max}) \leftarrow \text{element}(i,j,k,age_{max}) + \text{element}(i,j,k,age_{max-1}). \tag{4.18}$$

For example, to apply ageing to the categories `immature` and `mature`, then the syntax is,

```
@process Ageing
type ageing
categories immature mature
```

Note that ageing is *not* applied by `SPM` by default. As with other processes, `SPM` will not apply a process unless its defined and specified as a process within the annual cycle. Hence, it is possible to specify a model where a category is not aged. `SPM` will not check or otherwise warn if there is a category defined where ageing is not applied.

4.7.3. Mortality

Six types of mortality processes are permissible in `SPM`, constant rate, annually varying rate, event, biomass-event, a density-dependent relationship based on the Holling (Holling, 1959) Type II or Type III function, and a density-dependent relationship based on prey-suitability process. These processes remove individuals from the partition, either as a rate (for constant or annually varying mortality processes), as a total number (abundance), or as a biomass of individuals. `SPM` does not implement the Baranov catch equation or any other process where both natural and event mortality are applied simultaneously. To approximate concurrent natural and event mortality, the population processes must be defined to remove some natural mortality (e.g., as a constant or annually varying), then some event mortality (e.g., fishing) in sequence. It is up to the user to specify how this happens.

The constant rate, annually varying rate, event, biomass-event mortalities can depend on a layer. In these cases, the value of instantaneous mortality applied to the population state within each cell is the product of the layer value, a selectivity, and the mortality rate. If the layer is static, mortality is effectively constant each year but can be different in each cell; if the layer is a derived quantity and therefore calculated each year, mortality can change every year in every cell.

Constant mortality rate

To specify an constant annual mortality rate (i.e., $M = 0.2$) for categories 'male' and 'female', then,

```
@process NaturalMortality
type constant_mortality_rate
categories male female
selectivities One One
M 0.2 0.2
```

Note that the mortality rate process requires a selectivity. To apply the same mortality rate over all age classes, use a selectivity defined as $S_i = 1.0$ for all ages i, e.g.,

```
@selectivity One
type constant
c 1
```

A constant rate could also be defined as a multiplier of a layer. For example, let the mortality rate applied to the population at cell a in category k and age l be denoted $M(a,k,l)$, and given a value from a layer L_a at a, a constant mortality rate M, and a selectivity-at-age S_l at age l for some user-defined categories k then,

$$M(a,k,l) = ML_aS_l \tag{4.19}$$

And the resulting number of individuals remaining in cell a in category k at age l from applying the constant mortality process is,

$$n'(a,k,l) = n(a,k,l)\exp\left(-M(a,k,l)\right) \tag{4.20}$$

Constant exploitation rate

To specify an constant annual exploitation rate (i.e., $U = 0.2$) for categories 'male' and 'female', then,

```
@process ExploitationMortality
type constant_exploitation_rate
categories male female
selectivities One One
U 0.2 0.2
```

Note that the exploitation rate process requires a selectivity. To apply the same exploitation rate over all age classes, use a selectivity defined as $S_i = 1.0$ for all ages i, e.g.,

```
@selectivity One
type constant
c 1
```

A constant exploitation rate could also be defined as a multiplier of a layer. For example, let the exploitation rate applied to the population at cell a in category k and age l be denoted $U(a,k,l)$, and given a value from a layer L_a at a, a constant exploitation rate U, and a selectivity-at-age S_l at age l for some user-defined categories k then,

$$U(a,k,l) = UL_aS_l \tag{4.21}$$

with $0 <= U(a,l,k) <= U_{max}$.

And the resulting number of individuals remaining in cell a in category k at age l from applying the constant mortality process is,

$$n'(a,k,l) = n(a,k,l)U(a,k,l) \tag{4.22}$$

Annual mortality rate

Mortality for the annual rate is similar to the constant rate, except that rate applied each year is a separate parameter, and is applied equally to all of the specified categories. For example, if annual rates were applied to males and females between 1996 and 2000, five values of M and the years that these apply to would need to be provided, e.g.,

```
@process AnnualMortality
type annual_mortality_rate
categories male female
selectivities One One
years 1996 1997 1998 1999 2000
M 0.20 0.15 0.22 0.25 0.21
```

Event and biomass-event mortality

The event mortality process and biomass mortality processes act in a similar manner, except that they remove a specified abundance (number of individuals) or biomass respectively, rather than applying mortality as a rate. However, the maximum abundance or biomass to remove is constrained by a maximum exploitation rate.

The event mortality types must be defined using a layer. Here, the abundance or biomass to remove from a the population for each cell a is the value of the layer at a (denoted F_a) — except where there are too few individuals for the event mortality to be taken (as defined by the maximum exploitation rate). In this scenario, `SPM` removes as many individuals or as much biomass as it can while not exceeding the maximum exploitation rate. Event mortality processes require a penalty function to discourage parameter values that do not allow a the defined number of individuals to be removed. Here, the model penalises those parameter estimates that result in an insufficient number of individuals in defined categories (after applying selectivities). See Section 5.8 for more information on specifying penalties.

For example, the event mortality applied to user-defined categories k, with the numbers removed at age l determined by a selectivity-at-age S_l is applied as follows:

First, calculate the vulnerable abundance for each category k in $1 \ldots K$ for ages $l = 1 \ldots L$ that are subject to event mortality,

$$V(k,l) = S(l)N(k,l) \tag{4.23}$$

And hence define the total vulnerable abundance V_{total} as,

$$V_{total} = \sum_K \sum_L V(k,l) \tag{4.24}$$

Hence the exploitation rate to apply is

$$U = \begin{cases} C/V_{total}, & \text{if } C/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.25}$$

And the number removed R from each age l in category k is,

$$R(k,l) = UV(k,l) \tag{4.26}$$

For example, to specify fishing mortality based on spatially explicit catches (and given for each year as a layer, 'Catch2000', 'Catch2001', etc.) over categories 'immature' and 'mature' with selectivity 'FishingSel' and assuming a maximum possible exploitation rate of 0.7, then the syntax is,

```
@process Fishing
type event_mortality
categories immature mature
years 2000 2001 2002 2003
layers Catch2000 Catch2001 Catch2002 Catch2003
U_max 0.70
selectivities FishingSel FishingSel
penalty event_mortality_penalty
```

Holling mortality rate

The density-dependent Holling mortality process applies the Holling Type II and Type III functions (Holling, 1959), but is generalised using the Michaelis-Menten equation (Michaelis and Menten, 1913). The function removes a number or biomass from a set of categories according to their total (selected) abundance (or biomass)and some 'predator' abundance (or biomass), but constrained by a maximum exploitation rate.

For example, the mortality applied to user-defined categories k, with the numbers removed at age l determined by a selectivity-at-age $S(l)$ is applied as follows:

First, calculate the total predator abundance (or biomass) over all predator categories k in $1 \ldots K$ and ages $l = 1 \ldots L$ that are applying the mortality,

$$P(k,l) = S_{predator}(l) N_{predator}(k,l) \tag{4.27}$$

And define the total predator abundance (or biomass) P_{total} as,

$$P_{total} = \sum_K \sum_L P(k,l) \tag{4.28}$$

Then, calculate the total vulnerable abundance (or biomass) over all prey categories k in $1 \ldots K$ and ages $l = 1 \ldots L$ that are subject to the mortality,

$$V(k,l) = S_{prey}(l) N_{prey}(k,l) \tag{4.29}$$

And hence define the total vulnerable abundance (or biomass) V_{total} as,

$$V_{total} = \sum_K \sum_L V(k,l) \tag{4.30}$$

and then, the the number to remove is determined as,

$$R_{total} = P_{total} \frac{a V_{total}^{x-1}}{b + V_{total}^{x-1}} \tag{4.31}$$

where $x = 2$ for Holling type II function, $x = 3$ for Holling type III function, or any value of $x \geq 1$ for the generalised Michaelis-Menten function, and $a > 0$ and $b > 0$ are the Holling function parameters.

Hence the exploitation rate to apply is

$$U = \begin{cases} R_{total}/V_{total}, & \text{if } R_{total}/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.32}$$

And the number removed R from each age l in category k is,

$$R(k,l) = UV(k,l) \tag{4.33}$$

The density-dependent Holling mortality process is applied either as a biomass or an abundance depending on the value of the `is_abundance` switch.

For example, a biomass Holling type II mortality process on `prey` by our predator `predator` would have syntax,

```
@process HollingMortality
type Holling_mortality_rate
is_abundance F
a 0.08
b 10000
x 2
categories prey
selectivities One
predator_categories predator
predator_selectivities One
u_max 0.8
```

Prey-suitability mortality

The density-dependent prey-suitability process applies predation mortality from a predator group to its prey groups simultaneously. It removes an abundance (or biomass) from each prey group according to the total (selected) abundance (or biomass) of each prey group, the total (selected) abundance (or biomass) of the other prey groups, some 'predator' abundance (or biomass), and the preference (electivity) of the predator to each prey group, but constrained by a maximum exploitation rate. The predator-prey suitability functions were based on the multispecies Virtual Population Analysis (MSVPA) functions described by (Jurado-Molina et al., 2005).

For example, the mortality applied to the user-defined prey group g of category k, with the numbers removed at age l determined by a selectivity-at-age $S(l)$ is applied as follows:

First, calculate the total predator abundance (or biomass) over all predator categories k in $1 \ldots K$ and ages $l = 1 \ldots L$ that are applying the mortality,

$$P(k,l) = S_{predator}(l) N_{predator}(k,l) \tag{4.34}$$

And define the total predator abundance (or biomass) P_{total} as,

$$P_{total} = \sum_K \sum_L P(k,l) \tag{4.35}$$

Then, given the total vulnerable abundance (or biomass) of prey group g over all categories k in $1 \ldots K$ and ages $l = 1 \ldots L$ that are subject to the mortality,

$$V(g,k,l) = S_{prey}(l) N_{prey}(k,l) \tag{4.36}$$

And define the total vulnerable abundance (or biomass) of each prey group $V(g)_{total}$ as,

$$V(g)_{total} = \sum_K \sum_L V(g,k,l) \tag{4.37}$$

And the total availability $A(g)_{total}$ for each prey group as,

$$A(g)_{total} = \frac{V(g)_{total}}{\sum_G V(i)_{total}} \tag{4.38}$$

The vulnerable abundance (or biomass) and availability every prey group g in $1 \ldots G$ is calculated simultaneously. Then the abundance (or biomass) to remove from each prey group g is a function of its electivity $E(g)$, the availability of all other prey groups i in $1 \ldots G$, the electivity of the predator for each prey group $E(i)$, and the total consumption rate of the predator CR and its abundance (or biomass) P_{total},

$$R(g)_{total} = P_{total} CR \frac{A(g)_{total} E(g)}{\sum_G A(i)_{total} E(i)} \tag{4.39}$$

Hence the exploitation rate to apply to each prey group g is

$$U(g) = \begin{cases} R(g)_{total} / V(g)_{total}, & \text{if } R(g)_{total} / V(g)_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.40}$$

And the number removed $R(g)$ in each prey group g from each age l in category k is,

$$R(g,k,l) = U(g) V(g,k,l) \tag{4.41}$$

Note that prey suitability choice occurs only between prey groups specified by the process, and the total predator consumption rate represents the consumption of the predator on those prey groups alone. Also note that the electivities must sum to one. Further, the consumption rate can be modified by a layer, to be cell specific.

The density-dependent prey-suitability process is applied as either a biomass or an abundance depending on the value of the `is_abundance` switch.

Individual categories can be aggregated into prey groups using the '+' symbol. To indicate that two (or more) categories are to be aggregated, separate them with a '+' symbol. For example, to specify two prey groups of two species made up of the males and females in each, then the subcommand would be,

```
prey_categories maleSpeciesA + femaleSpeciesA maleSpeciesB + femaleSpeciesB
```

This would indicate that there are two prey groups, `maleSpeciesA + femaleSpeciesA` and `maleSpeciesB + femaleSpeciesB`, with each group having its own electivity. For example, a biomass prey-suitability mortality process with an overall consumption rate of 0.8 of `species A` and `species B` (modelled as males and females) by our predator `predatorSpecies` with electivities between `species A` and `species B` of 0.18 and 0.82 would have syntax,

```
@process PreySuitabilityMortality
type prey-suitability_predation
is_abundance F
consumption_rate 0.8
categories maleSpeciesA + femaleSpeciesA maleSpeciesB + femaleSpeciesB
electivities 0.18 0.82
selectivities One One One One
predator_categories predatorSpecies
predator_selectivities One
u_max 0.8
```

4.7.4. Category states

Category state processes set the number of individuals at each age and category at a specified point in time during the model. `SPM` implements only one type, the number in each age class and category in each cell (category state by age).

Category state by age process

The category state by age process sets the number n in each age for a single category in each cell. This process may be used, for example, to initialise the partition at the start of a model for simulations or other scenario testing. The process for category a and age l in cell(i, j) is,

$$\text{element}(i, j, a, l) \leftarrow N(i, j, a, l) \tag{4.42}$$

Note that this process will overwrite the information that may already exist within a spatial cell — cells that are not defined will be left with their original values. Similarly, this process will overwrite the information that may already exist within the defined age classes in the specified category — age classes that are not defined will be left with their original values.

Only a single category can be defined — use multiple processes to specify multiple categories. For example, to initialise the immature category in the partition, then you might define the category state processes `immature` as a process to run during an initialisation phase, with the process defined as,

```
@process initialise-immature
type category_state_by_age
category immature
layer Area
min_age 1
max_age 5
N A 2 5 15 8 12
N B 2 5 20 20 8
```

where the layer specifies how the number of individuals are assigned to individual cells. In this example, assuming a 2×2 spatial model with the categorical layer (e.g., with label `Area`) defined as,

```
@layer Area
type categorical
data A A
data B B
```

then the state in the initialisation with label `phase1` would be set to,

```
cell  age=1 age=2 age=3 age=4 age=5 age=6
r1-c1     1   2.5   7.5     4     6     0
r1-c2     1   2.5   7.5     4     6     0
r2-c1     1   2.5    10    10     4     0
r2-c2     1   2.5    10    10     4     0
```

assuming that there were no age 6 individuals previously in the partition.

4.7.5. Category transitions

Category transition processes move individuals between categories. `SPM` implements three types, the total number in each cell (category transition), a rate in each cell (category transition rate), and the number in each age class in each cell (category transition by age).

Category transition process

The category transition process moves a number n from some source to a sink category (or categories). This process may be used, for example, to implement a 'tagging' process for mark-recapture data. The transition process with selectivity S for source category a and sink category b (in cell(i,j) and age l) is,

$$\begin{aligned} \text{element}(i,j,a,l) &\leftarrow \text{element}(i,j,a,l) - \frac{nS_l}{\sum_l S_l} \times \text{element}(i,j,a,l) \\ \text{element}(i,j,b,l) &\leftarrow \text{element}(i,j,b,l) + \frac{nS_l}{\sum_l S_l} \times \text{element}(i,j,a,l) \end{aligned} \tag{4.43}$$

The category transition process requires a penalty function to discourage parameter values that do not allow the specified number of individuals to be moved. Here, the model penalises those parameter estimates that result in an insufficient number of individuals in the source category (after applying the selectivity) available to be moved. See Section 5.8 for more information on specifying penalties.

If multiple categories of sources or sinks are defined, then they must be defined in 'pairs'. The proportions of selected individuals in the source categories is used to define the proportions of individuals 'moved' to the sink categories, as defined by the order that they are specified. For example, to 'tag' a population of immature and mature individuals, then you might define a category transition process `tagging` with selectivities `tagging-Sel`, as,

```
@process tagging
type category_transition
from immature mature
selectivities tagging-Sel tagging-Sel
to immature-tag mature-tag
years 2001 2002 2003
layers TagRelease2001 TagRelease2002 TagRelease2003
penalty tag_release_penalty
```

Note that this syntax can be used to combine individuals into categories, simply by repeating a category label when specifying the sink categories. Similarly, individuals within a category can be split into more than one category by repeating a category label when specifying the source categories.

Category transition rate process

The category transition rate process moves a proportion p from a source category to a and sink category (or categories). The category transition rate process with selectivity S for source category a and sink category b (in cell(i,j) and age l) is,

$$\begin{aligned} \text{element}(i,j,a,l) &\leftarrow \text{element}(i,j,a,l) - pS_l \times \text{element}(i,j,a,l) \\ \text{element}(i,j,b,l) &\leftarrow \text{element}(i,j,b,l) + pS_l \times \text{element}(i,j,a,l) \end{aligned} \tag{4.44}$$

If multiple categories of sources or sinks are defined, then they must be defined in 'pairs'. `SPM` treats each pair of categories as an independent transition — but note that these are applied in order that they are specified. For example, to 'mature' males and females in a model with four categories `male-immature`, `female-immature`, `male-mature`, and `female-mature`, then you might define a category transition process `maturation` with selectivities `male-maturity` and `female-maturity`, as,

```
@process maturation
type category_transition_rate
from male-immature female-immature
to male-mature female-mature
proportions 1.0 1.0
selectivities male-maturity female-maturity
```

Note that this syntax can be used to combine individuals into categories, simply by repeating a category label when specifying the sink categories. Similarly, individuals within a category can be split into more than one category by repeating a category label when specifying the source categories.

Category transition by age process

The category transition by age process moves a number n for each age class from some source to a sink category (or categories) in a given year in subsets of cells defined by a categorical layer. This process may be used, for example, to implement a 'tagging' process for mark-recapture data, were the number of individuals tagged at each age are known. If the process is to be applied over more than one year, multiple category transition processes by age will need to be defined — one for each year. The transition process with selectivity S for source category a and sink category b (in cell(i,j) at age l) is,

$$\begin{aligned} \text{element}(i,j,a,l) &\leftarrow \text{element}(i,j,a,l) - \frac{nS_l}{\sum_l S_l} \times \text{element}(i,j,a,l) \\ \text{element}(i,j,b,l) &\leftarrow \text{element}(i,j,b,l) + \frac{nS_l}{\sum_l S_l} \times \text{element}(i,j,a,l) \end{aligned} \tag{4.45}$$

Category transition by age processes require a penalty function to discourage parameter values that do not allow the specified number of individuals to be moved. Here, the model penalises those parameter estimates that result in an insufficient number of individuals in the source category (after applying the selectivity) available to be moved. See Section 5.8 for more information on specifying penalties.

If multiple categories of sources or sinks are defined, then they must be defined in 'pairs'. The proportions of selected individuals in the source categories is used to define the proportions of individuals 'moved' to the sink categories, as defined by the order that they are specified. For example, to 'tag' a population of immature and mature individuals, then you might define a category transition process `tagging` with selectivities `tagging-Sel`, as,

For example, in a 2×2 spatial model a categorical layer (e.g., with label Area) may define that cells $(1,1)$ and $(1,2)$ have value A and cells $(2,1)$ and $(2,2)$ have value B, i.e.,

```
@layer Area
type categorical
data A A
data B B
```

Then the category transition by age process that moves 40 individuals in the subset of cells with label A and 55 individuals in the subset of cells with label B may be defined as,

```
@process tagging
type category_transition_by_age
from immature mature
to immature-tag mature-tag
layer Area
year 2001
min_age 1
max_age 5
N A 1 5 15 8 11
N B 2 5 20 20 8
penalty tag_release_penalty
```

Note that this syntax can be used to combine individuals into categories, simply by repeating a category label when specifying the sink categories. Similarly, individuals within a category can be split into more than one category by repeating a category label when specifying the source categories.

4.8. Movement processes

Movement processes are those processes that move individuals between cells but retain their population state, and are defined such that,

$$\text{element}(i,j,k,l) \leftarrow \text{element}(i,j,k,l) + p \times \text{element}(i',j',k,l) \tag{4.46}$$

i.e., each element in cell(i,j) is updated as the sum of itself and some proportion p of a neighbouring element in cell(i',j'). To conserve abundance we also update element(i',j',k,l) as,

$$\text{element}(i',j',k,l) \leftarrow \text{element}(i',j',k,l) - p \times \text{element}(i',j',k,l) \tag{4.47}$$

`SPM` assumes that each movement process occurs simultaneously over all cells (synchronous updating), i.e., all cell updates from each individual movement process are first evaluated for all cells, and then applied to all cells affected.

SPM implements three types of movement;

1. A migration movement rate of cohorts between any two locations, and is roughly analogous to movements between areas as implemented in other population models, such as CASAL (Bull et al., 2012).
2. An adjacent cell movements, parametrised by some function of an underlying layer — equivalent to, for example, movement processes implemented in Fish Heaven (Ball and Constable, 2000, Ball and Williamson, 2003).
3. Movement parametrised as a probability density function. Here, the key underlying idea is that the spatial distribution of cohorts at any point in time and at any location can be represented as a density function based on attributes of that location, local abundance, and/or distance from their previous location (Bentley et al., 2004a,b).

4.8.1. Migration movement

The migration process moves individuals from one sets of locations (the source, or emigration layer) to another set of locations (the sink, or immigration layer). A migration can involve one or more categories. The emigration at age is defined as some constant proportion multiplied by a selectivity and the source layer applied as a proportion. The immigration at age is defined as the number emigrating at age distributed proportionally into the sink layer. Migrations are similar to the migration process used in some limited space models such as CASAL (Bull et al., 2012).

For example, let the emigration applied to the population at cell a in category k and age l be denoted $E(a,k,l)$, and given a value from an source layer Le_a at a, a constant migration proportion P, and a selectivity-at-age S_l at age l for some user-defined categories k then,

$$E(a,k,l) = \frac{PLe_aS_l}{\sum_a Le_a} \tag{4.48}$$

And let the immigration applied to the population at cell a in category k and age l be denoted $I(a,k,l)$, and given a value from an sink layer Li_a at a, for some user-defined categories k then,

$$I(a,k,l) = \frac{E(a,k,l)Li_a}{\sum_a Li_a} \tag{4.49}$$

For example, to migrate 50% of males from the top left cell to the three cells in the bottom right of a 3×3 model, then first define the layers to define the movement source and sinks,

```
@layer source
type numeric
data 1 0 0
data 0 0 0
data 0 0 0

@layer sink
type numeric
data 0 0 0
data 0 0 1
data 0 1 1
```

And then the migration is defined as,

```
@process MaleMigration
type migration
categories males
proportion 0.5
source_layer source
sink_layer sink
selectivities one
```

4.8.2. Adjacent cell movement

The adjacent cell movement moves a proportion of individuals in each cell to its four neighbouring cells, and hence mimics a simple diffusion process. It can be applied to a limited range of spatial locations and/or as a gradient by using a layer. An adjacent cell movement can involve one or more categories and movement at age is defined as some constant proportion multiplied by a selectivity and the diffusion layer as a proportion over the four adjoining cells. If no layer is supplied, the process moves the population equally to the adjacent cells (i.e., a constant 0.25 proportion is assumed for each of the adjacent cells).

For example, let the movement from cell a to neighbouring cell b in category k and age l be denoted $V(a,b,k,l)$, and given a value from layer L_b at b, values from layer L_n at the four neighbouring cells to a (including cell b), a constant migration proportion P, and a selectivity-at-age S_l at age l for some user-defined categories k then,

$$V(a,b,k,l) = \frac{PL_bS_l}{\sum_4 L_n} \tag{4.50}$$

The movement value of all cells into all their neighbouring cells are calculated simultaneously from the population layer at the time of movement, and the balance of all movements is returned.

For example, to move 50% of females to adjacent cells but favouring movement towards the bottom right cells, then first define a gradient layer,

```
@layer diffusion_layer
type numeric
data 1 1 1
data 1 2 2
data 1 2 4
```

And then the movement is defined as,

```
@process diffusion
type adjacent_cell
categories female
proportion 0.5
layer diffusion_layer
selectivities one
```

4.8.3. Preference movement

Preference movements allows movement from any $cell(a) \rightarrow cell(b)$, for $\forall a,b \in L_B$ and is implemented as a function of the product of up to n independent *preference functions*. We define

the probability of moving from any cell a to any cell b, for all $a, b \in L_B$, as a function of the relative preference for that cell. Here, we use the term *preference function* (Bentley et al., 2004a,b) to describe the movement probability distributions.

We assume that the population and spatial extent are defined, and that there is a preference function that is a function of some (typically estimable) parameters and a spatially explicit set of known attributes.The preference function movement process allows the number of parameters describing movement to to reduced, and results in a movement process that is some function of some underlying property of each location. For example, if we assume that movement between areas was a function of the Euclidean distance between areas, we could model movement between any two areas as a linear decay or exponential decay function (Bentley et al., 2004a). Alternately, if distribution and density were correlated with bathymetric depth for a marine organism, we might model the movement and distribution as a function of depth.

Note that two forms of the preference movement algorithm are implemented in `SPM`. They implement the same algorithm, except that the `@process.type=preference_threaded` implements movement as a multi-threaded CPU process (allowing multiple between cell movements to be evaluated by the computer simultaneously) and `@process.type=preference` implements movement as a single-threaded CPU process. The simultaneous evaluation of movements between cells in the multi-threaded movement process can result in a significantly faster processing time for an individual model (though, due to the thread management overhead, only for large models). Note that a consequence of this is that the order of calculations for the multi-thread and single-thread processes *may* differ in some cases, and that this may result in a slightly different outcome.

The total preference function

Movement in `SPM` can be defined as a probability distribution based on an underlying preference function. Here, we define the preference for a cell x as the preference function $f_x(\theta_x, P(x))$, where θ_x are the parameters for f_x. So, given a set of n attributes for cell x, we can define a preference function for each, and hence we define the aggregated or total preference function for any cell x as the weighted product of individual preference functions,

$$P_x = f_1(\theta_1, P_1(x))^{\alpha_1} \times f_2(\theta_2, P_2(x))^{\alpha_2} \times f_3(\theta_3, P_3(x))^{\alpha_3} \times \cdots \times f_n(\theta_n, P_n(x))^{\alpha_n} \tag{4.51}$$

where α_i is an arbitrary weighting factor for attribute i. In order to avoid over-parameterisation, it is recommended that at least one α_i be fixed to the value of one.

Then we define the probability of moving from cell a to any cell b (where b is defined as the set of all possible cells, including a),

$$p(a \to b) = \frac{P_a}{\sum_{i \in \forall b} P_i} \tag{4.52}$$

Note that there are three forms of preference function,

1. Those that are a function of some underlying attribute of a cell, as defined by some arbitrary layer L
2. Those that are a function of the abundance or biomass (perhaps with a selectivity and for a subset of all categories) of each cell
3. Those that are a function of the distance between the sink and the source cells.

Preference functions of the first type are determined only by the parameters of the preference function and some underlying, fixed, attribute. Preference functions of the others are dynamic, i.e. they depend on the relative locations of the cells or on the density of a cell at a particular point in time.

SPMcan also incorporate dependence between two variables as a preference function using the copula method (see later).

Preference functions

Preference functions in SPM include constant, Normal, double-Normal, logistic, inverse-logistic, Exponential, threshold, categorical, and monotonic categorical. These are defined as,

1. The constant preference function has dependent variable x and has no parameters, and is defined as,

$$f(x) = x, \text{ where } 0 \leq x \leq 1 \tag{4.53}$$

2. The Normal preference function has dependent variable x and parameters $\theta = (\mu, \sigma)$, and is defined as,

$$f(x|\mu, \sigma) = 2^{-[(x-\mu)/\sigma]^2} \tag{4.54}$$

3. The double-Normal preference function has dependent variable x and parameters $\theta = (\mu, \sigma_L, \sigma_R)$, and is defined as,

$$f(x|\mu, \sigma_L, \sigma_R) = \begin{cases} 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \tag{4.55}$$

4. The Logistic preference function has dependent variable x and parameters $\theta = (a_{50}, a_{to95})$, and is defined as,

$$f(x|a_{50}, a_{to95}) = 1/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.56}$$

5. The inverse-Logistic preference function has dependent variable x and parameters $\theta = (a_{50}, a_{to95})$, and is defined as,

$$f(x|a_{50}, a_{to95}) = 1 - 1/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.57}$$

6. The Exponential preference function has dependent variable x and parameter $\theta = (\lambda)$, and is defined as,

$$f(x|\lambda) = \exp(-\lambda x), \text{where } x \geq 0 \text{ and } 0 \text{ otherwise} \tag{4.58}$$

7. The threshold preference function has dependent variable x and parameters $\theta = (N, \lambda)$, and is defined as,

$$f(x|N, \lambda) = \begin{cases} 1, & \text{if } 0 \leq x \leq N \\ 1/\left(\frac{x}{N}^{\lambda}\right), & \text{if } x \geq N \\ 0, & \text{otherwise} \end{cases} \tag{4.59}$$

8. The categorical preference function has dependent variables x_i and parameters $\theta = (\lambda_i)$, and is defined so that for each value x_i, there is a corresponding parameter λ_i.

   Note that for the categorical preference function, the preference function is potentially over-parameterised if $\alpha \neq 1$. Typically the values of x are supplied via a categorical layer, with x_i representing the unique values of the layer.

9. The monotonic categorical preference function has dependent variables x_i and parameters $\theta = (\hat{\lambda}_i)$. As for the categorical preference function, it is defined so that for each x_i, there is a corresponding parameter $\hat{\lambda}_i = \lambda_i$ for the first unique value of x_i and $\hat{\lambda}_i = \lambda_i + \lambda_{i-1}$ otherwise, and that $\forall i, \lambda_i \geq = 0$.

   Note that for the monotonic categorical preference function, the preference function is potentially over-parameterised if $\alpha \neq 1$. Typically the values of x are supplied via a categorical layer, with x_i representing the unique values of the layer.

For example, to define a preference based movement that is the combination of a preferred depth (assumed to be given in the layer `Depth`) and distance (given in, say, the layer `Distance`, first define the individual preference functions,

```
@preference_function Depth
type double_normal
alpha 1
layer Depth
mu 600
sigma_l 120
sigma_r 150

@preference_function Distance
type exponential
alpha 1
lambda 0.005
layer Distance
```

And then the preference based movement process is simply,

```
@process Move
type preference
categories males+females
preference_functions Distance Depth
```

Copula Movement

The dependence between two variables can be included as a preference function using the copula method. This uses marginal Probability Distribution Functions (PDF) along with an assumed dependence structure to calculate the joint preference distribution between two preference variables (Marsh, 2015). Here, the joint distribution for the two variables x_1 and x_2 with known marginal PDF is given by,

$$f_{x_1,x_2}(x_1,x_2;u_1,u_2,\phi_1,\phi_2,\rho) = c(u_1,u_2;\rho)f_1(x_1;\phi_1)f_1(x_1;\phi_1) \tag{4.60}$$

where, $c(u_1,u_2;\rho)$ is a copula density representing the structure of dependence between X_1 and X_2 with $0 \leq (u_1,u_2) \leq 1$, and $f_{x_1,x_2}(x_1,x_2;u_1,u_2,\phi_1,\phi_2,\rho)$ is the bivariate joint distribution of X_1 and X_2, and $u_i = F_i(x_i;\phi_i)$ is the cumulative distribution function from the corresponding PDF.

The marginal probability distributions available are,

1. The normal distribution has variable x and parameters $\phi = (\mu, \sigma)$, and is defined as,

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp -(\frac{(x-\mu)^2}{2\sigma^2}) \tag{4.61}$$

2. The lognormal distribution has variable x and parameters $\phi = (\mu, \sigma)$, and is defined as,

$$f(x;\mu,\sigma) = \frac{1}{x_i\sigma\sqrt{2\pi}} \exp -(\frac{(\log x-\mu)^2}{2\sigma^2}) \tag{4.62}$$

3. The exponential distribution has variable x and parameters $\phi = (\lambda)$, and is defined as,

$$f(x;\lambda) = \lambda \exp(-\lambda x) \tag{4.63}$$

4. The uniform distribution has variable x and parameters $\phi = (a, b)$, and is defined as,

$$f(x;a,b) = \frac{1}{b-a} \tag{4.64}$$

A marginal distribution must be chosen for each of the two preference variables when using the copula method. The dependence structures available are,

1. The Gaussian copula, which has variables x_1 and x_2 and parameter ρ, and is defined as,

$$c(.) = \frac{1}{2\pi\sqrt{1-\rho}} \exp(-\frac{\zeta_1^2 - 2\rho\zeta_1\zeta_2 + \zeta_2^2}{2(1-\rho^2)}) \tag{4.65}$$

   where $\rho \in [-1, 1]$

2. The Gumbel copula, which has variables x_1 and x_2 and parameter ρ, and is defined as,

$$\begin{aligned} c(.) &= \exp\{-[(-ln(u_1))^\rho + (-ln(u_2))^\rho]^{1/\rho}\} \frac{(-ln(u_1))^\theta - 1}{u_1} \frac{(-ln(u_2))^\rho - 1}{u_2} \\ &\quad \{[(-ln(u_1))^\rho + (-ln(u_2))^\rho]^{\frac{2}{\rho}-2} + (\rho - 1) \\ &\quad [(-ln(u_1))^\rho + (-ln(u_2))^\rho]^{\frac{1}{\theta}-2}\} \end{aligned} \tag{4.66}$$

   where $\rho \in [1, \infty)$

3. The Frank copula, which has variables x_1 and x_2 and parameter ρ, and is defined as,

$$\begin{aligned} c(.) &= \rho \exp\{-\rho u_1\} \exp\{-\rho u_2\} [(\exp\{-\rho\} - 1) + (\exp\{-\rho u_1\} - 1)(\exp\{-\rho u_2\} - 1)]^{-1} \\ &\quad \{(\exp\{-\rho u_1\} - 1)(\exp\{-\rho u_2\} - 1)((\exp\{-\rho\} - 1) \\ &\quad + (\exp\{-\rho u_1\} - 1)(\exp\{-\rho u_2\} - 1))^{-1} - 1\} \end{aligned} \tag{4.67}$$

   where $\rho! = 0$.

4. Independence copula, which has variables x_1 and x_2, and is defined as,

$$c(.) = f_1(x_1) f_2(x_2) \tag{4.68}$$

Dependence is described by quadrant dependence. Here, let X_1 and X_2 be random variables. Then X_1 and X_2 are positively quadrant dependent (PDQ) if for all (x_1, x_2) in R^2

$$P[X_1 \leq x_1, X_2 \leq x_2] \geq P[X_1 \leq x_1], P[X_2 \leq X_2] \tag{4.69}$$

or,

$$P[X_1 > x_1, X_2 > x_2] \geq P[X_1 > x_1], P[X_2 > x_2] \tag{4.70}$$

where R^2 is the two dimensional space of real numbers. Negative quadrant dependence (NQD) is defined analogously by reversing the inequalities. PQD says the probability that random variables are jointly large is greater than when they are looked at independently.

independence copula represents lack of dependence.

The Gaussian copula is an elliptical copula that captures the full range of positive and negative dependence.

The Frank copula can describe both PQD and NQD. When $\rho > 0$ it displays PQD, and when $\rho < 0$ it displays NQD.

The Gumbel copula cannot account for negative dependence but is well suited for the case when there is strong right tail dependence (strong association at high values) but weak left tail dependence (weak association at low values).

For example, an implementation of the Gumbel copula with normal and exponential marginal distributions would be defined as,

```
@preference_function SST_NPP
type gumbel_copula
rho 2
pdfs sst npp
layers sst npp

@pdf sst
type normal
mu 18
sigma 2

@pdf npp
type exponential
mu 1
sigma 500
```

4.9. Derived quantities

Some processes require, as arguments, a population value derived from the population state. These are termed *derived quantities*. Derived quantities are values, calculated by `SPM` as the end of a specified time-step in every year, and hence they have a single value for each year of the model. Derived quantities can be calculated as either an abundance or as a biomass. Abundance derived quantities are simply the count or sum of cells within some categories (after applying a selectivity) within cells defined by a layer. Biomass derived quantities are similar, except they are a measure of biomass. Derived quantities are also calculated during the initialisation phases, and hence the time-step during each phase must also be specified. If the initialisation time-steps are not specified,

SPM will calculate the derived quantity during the initialisation phases in every year, at the end of the annual cycle.

Derived quantities are required by some processes, for example the Beverton-Holt recruitment process. The Beverton-Holt recruitment process requires an equilibrium biomass (B_0) and annual spawning stock biomass values (SSB_y) to resolve the stock-recruit relationship. Here, these would be defined as the abundance or biomass of a part of the population at some point in the annual cycle for selected ages and categories, and would be calculated as a derived quantity.

As an example, to define a biomass derived quantity (say spawning stock biomass, SSB) for a model, evaluated at the end of the first time-step (labelled step_one) in the initialisation phases (and assuming two initialisation phases) and at the end of the first time-step otherwise, over areas defined by a layer as the spawning ground but counting all 'mature' individuals, we would use the syntax,

```
@derived_quantity SSB
type biomass
time_step step_one
initialisation_time_steps step_one step_one
categories mature
selectivities One
layer spawning_ground
```

4.10. Derived quantities by cell

Some processes require, as arguments, a population value derived from the population state, but available for each cell of the population. These are termed *derived quantities by cell*. Derived quantities by cell are a matrix of values, calculated by SPM as the end of a specified time-step in every year, and hence they have a single value for each cell in each year of the model. Derived quantities by cell can be calculated as either an abundance or as a biomass. Abundance derived quantities by cell are simply the count or sum within each cell for some categories (after applying a selectivity). Biomass quantities by cell layers are similar, except they are a measure of biomass. Derived quantities by cell are also calculated during the initialisation phases, and hence the time-step during each phase must also be specified. If the initialisation time-steps are not specified, SPM will calculate the derived quantity by cell during the initialisation phases in every year, at the end of the annual cycle.

Derived quantities by cell are *required* by some processes, for example the local Beverton-Holt recruitment process. The local Beverton-Holt recruitment process requires an equilibrium biomass (B_0) and annual spawning stock biomass values (SSB_y) in each cell to resolve the stock-recruit relationship for each cell. Here, these would be defined as the abundance or biomass of a part of the population at some point in the annual cycle for selected ages and categories, and would be calculated as a derived quantity by cell.

As an example, to define a biomass derived quantity by cell (say spawning stock biomass, SSB) for a model, evaluated at the end of the first time-step (labelled step_one) in the initialisation phases (and assuming two initialisation phases) and at the end of the first time-step otherwise, but counting all 'mature' individuals, we would use the syntax,

```
@derived_quantity_by_cell SSB
type biomass
time_step step_one
initialisation_time_steps step_one step_one
categories mature
selectivities One
```

4.11. Size-age relationship

The age-size relationship defines the size at age (and the weight at size, see Section 4.12) of individuals at age/category within the model. There are three size-age relationships available in SPM. The first is the naive no relationship (where each individual has size 1 irrespective of age). The second and third are the von-Bertalanffy and Schnute relationships respectively. The size-at-age relationship is used to determine the size frequency, given age, and then with the size-weight relationship, a weight-at-age of individuals within an age/category.

The three age-size relationships are,

None: where the size of each individual is exactly 1 for all ages, in which case the none size-weight relationship must also be used.

von Bertalanffy: where size at age is defined as,

$$\bar{s}(age) = L_\infty \left(1 - \exp\left(-k\left(age - t_0\right)\right)\right) \tag{4.71}$$

Schnute: where size at age is defined as,

$$\bar{s}(age) = \begin{cases} \left[y_1^b + (y_2^b - y_1^b)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right]^{1/b}, & \text{if } a \neq 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right], & \text{if } a \neq 0 \text{ and } b = 0 \\ \left[y_1^b + \left(y_2^b - y_1^b\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right]^{1/b}, & \text{if } a = 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right], & \text{if } a = 0 \text{ and } b = 0 \end{cases} \tag{4.72}$$

The von Bertalanffy curve is parameterised by L_∞, k, and t_0; the Schnute curve (Schnute, 1981) by y_1 and y_2, which are the mean sizes at reference ages τ_1 and τ_2, and a and b (when $b = 1$, this reduces to the von Bertalanffy with $k = a$).

When defining size-at-age in SPM, you must also define a size-weight relationship (see Section 4.12 below).

For example to define the age-at-size with label VB as a von-Bertalanffy relationship with $L_\infty = 100$, $k = 0.3$, and $t_0 = -0.1$, and a size-weight relationship (see below) given by SizeWeight,

```
@age_size VB
type von_bertalanffy
linf 100
k 0.3
t0 -0.1
size_weight SizeWeight
```

4.12. Size-weight relationship

There are two size-weight relationship,s available in SPM. The first is the naive no relationship. Here, the weight of an individual, regardless of size, is always 1. The second is the basic relationship.

The two size-weight relationships are,

- None: The size-weight relationship where

$$\text{mean weight} = 1 \tag{4.73}$$

- Basic: The size-weight relationship where the mean weight w of an individual of size l is

$$w = al^b \tag{4.74}$$

Note that if a distribution of size-at-age is specified, then the mean weight is calculated over the distribution of sizes, and is

$$w = (al^b)(1+cv^2)^{\frac{b(b-1)}{2}} \tag{4.75}$$

where the cv is the c.v. of sizes-at-age. This adjustment is exact for lognormal distributions, and a close approximation for normal distributions if the c.v. is not large (Bull et al., 2012).

Be careful about the scale of a — this can easily be specified incorrectly. If the catch is in tonnes and the growth curve in centimetres, then a should be on the right scale to convert a length in centimetres to a weight in tonnes. Note that there are reports available that can be used to help check that the units specified are plausible (see Section 7).

For example to define the size-weight with label `SizeWeight` as the basic relationship with $a = 1.4e-8$ and $b = 3$,

```
@size_weight SizeWeight
type basic
a 1.4e-008
b 3
```

4.13. Selectivities

A selectivity is a function with a different value for each age class (i.e., for each column of the partition). Selectivities are used throughout `SPM` to interpret observations (Section 5) or to modify the effects of processes on each age class (Section 4). `SPM` implements a number of different parametric forms, including logistic, knife edge, and double normal selectivities.

A selectivity is always defined to apply just to one category of the population (i.e, row of the partition). To apply the same selectivity to more than one category, then just repeat the selectivity for each category that it is applied to.

Note that selectivities are indexed by age, with indices from `min_age` to `max_age`. For example, you might have an age-based selectivity that was logistic with 50% selected at age 5 and 95% selected at age 7. This would be defined by the `type=logistic` with parameters $a_{50} = 5$ and $a_{to95} = (7-5) = 2$. Then the value of the selectivity at age $x = 7$ is 0.95 and the selectivity at $x = 3$ is 0.05.

Note that the function values for some choices of parameters for some selectivities can result in an computer numeric overflow error (i.e., the number calculated from parameter values is either too large or too small to be represented in computer memory). `SPM` implements range checks on some parameters to test for a possible numeric overflow error before attempting to calculate function values. For example, the logistic selectivity is implemented such that if $(a50-x)/ato_95 > 5$ then the value of the selectivity at $x = 0$, i.e., for $a50 = 5$, $ato_95 = 0.1$, then the value of the selectivity at $x = 1$, without range checking would be 7.1×10^{-52}. With range checking, that value is 0 (as $(a50x)/ato_95 = 40 > 5$).

The available selectivities are;

- Constant
- Knife-edge
- All values
- All values bounded
- Increasing
- Logistic
- Inverse logistic
- Logistic producing
- Double normal
- Double exponential
- Cubic spline

The available selectivities are described below.

4.13.1. `constant`

$$f(x) = C \tag{4.76}$$

The constant selectivity has the estimable parameter C.

For example, to define a constant selectivity with $c = 1$, use,

```
@selectivity Sel
type constant
c 1
```

4.13.2. `knife_edge`

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{4.77}$$

The knife-edge selectivity has the estimable parameter $E$and a scaling parameter α, where the default value of $\alpha = 1$

For example, to define a knife-edge selectivity with $= 12$, use,

```
@selectivity Sel
type knife_edge
e 12
```

4.13.3. `all_values`

$$f(x) = V_x \tag{4.78}$$

The all-values selectivity has estimable parameters V_{low}, $V_{low+1} \ldots V_{high}$. Here, you need to provide the selectivity value for each age class.

For example, to define an all-values selectivity for a model with ages $1 \ldots 6$, and the selectivity at age linearly increasing from 0 to 1 over those ages, use

```
@selectivity Sel
type all_values
v  0.0 0.2 0.4 0.6 0.8 1.0
```

4.13.4. all_values_bounded

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \le x \le H \\ V_H, & \text{if } x > H \end{cases} \tag{4.79}$$

The all-values-bounded selectivity has non-estimable parameters L and H. The estimable parameters are V_L, V_{L+1} ... V_H. Here, you need to provide an selectivity value for each age class from $L \ldots H$.

For example, to define an all-values-bounded selectivity for a model with ages 1 to 6, and the selectivity at age linearly increasing from 0 to 1 over ages $1 \ldots 4$ and constant thereafter, use

```
@selectivity Sel
type all_values_bounded
l 1
h 4
v 0.0 0.33 0.66 1.0
```

4.13.5. increasing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \le x \le H \\ f(\alpha), & \text{if } x \ge H \end{cases} \tag{4.80}$$

The increasing ogive has non-estimable parameters L and H. The estimable parameters are π_L, π_{L+1} ... π_H (but if these are estimated, they should always be constrained to be between 0 and 1). α is a scaling parameter, with default value of $\alpha = 1$. Note that the increasing ogive is similar to the all-values-bounded ogive, but is constrained to be non-decreasing.

For example, to define an increasing selectivity for a model with ages 1 to 6, and the selectivity at age increasing by 0.1 from 0.5 over ages $1 \ldots 4$ and constant thereafter, use

```
@selectivity Sel
type increasing
l 1
h 4
v 0.5 0.1 0.1 1.0
```

4.13.6. logistic

$$f(x) = \alpha / [1 + 19^{(a_{50} - x)/a_{to95}}] \tag{4.81}$$

The logistic selectivity has estimable parameters a_{50} and a_{to95}. α is a scaling parameter, with default value of $\alpha = 1$. The logistic selectivity takes values 0.5α at $x = a_{50}$ and 0.95α at $x = a_{50} + a_{to95}$.

For example, to define a logistic selectivity for a model with ages 1 to 6 with $a_{50} = 3$ and $a_{to95} = 1$, with a maximum selectivity of 1.0, use

```
@selectivity Sel
type logistic
alpha 1
a50 4
ato95 1
```

4.13.7. inverse_logistic

Selectivities!Inverse-logistic

$$f(x) = \alpha - \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.82}$$

The inverse-logistic selectivity has estimable parameters a_{50} and a_{to95}. α is a scaling parameter, with default value of $\alpha = 1$. The logistic selectivity takes values 0.5α at $x = a_{50}$ and 0.95α at $x = a_{50} - a_{to95}$.

For example, to define an inverse-logistic selectivity for a model with ages 1 to 6 with $a_{50} = 3$ and $a_{to95} = 1$, with a maximum selectivity of 1.0, use

```
@selectivity Sel
type inverse_logistic
alpha 1
a50 4
ato95 1
```

4.13.8. logistic_producing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1)) / (1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \tag{4.83}$$

The logistic-producing selectivity has the non-estimable parameters L and H, and has estimable parameters a_{50} and a_{to95}. α is a scaling parameter, with default value of $\alpha = 1$. For category transitions, $f(x)$ represents the proportion moving, not the proportion that have moved. This selectivity was designed for use in an age-based model to model maturity. In such a model, a logistic-producing maturation selectivity will (in the absence of other influences) make the proportions mature follow a logistic curve with parameters a_{50}, a_{to95}.

For example, to define an logistic-producing selectivity for a model with ages 1 to 6 with $a_{50} = 3$ and $a_{to95} = 1$, with a maximum selectivity of 1.0, use

```
@selectivity Sel
type logistic_producing
alpha 1
l 1
h 6
a50 4
ato95 1
```

4.13.9. double_normal

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \le \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \ge \mu \end{cases} \tag{4.84}$$

The double-normal selectivity has estimable parameters a_1, s_L, and s_R. α is a scaling parameter, with default value of $\alpha = 1$. It has values α at $x = a_1$, and 0.5α at $x = a_1 - s_L$ and $x = a_1 + s_R$.

For example, to define an double-normal selectivity for a model with ages 1 to 6 with $\mu = 3$, $\sigma_L = 1$, and $\sigma_R = 1$, with a maximum selectivity of 1.0, use

```
@selectivity Sel
type double_normal
alpha 1
mu 3
sigma_l 1
sigma_r 1
```

4.13.10. double_exponential

$$f(x) = \begin{cases} \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \le x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{4.85}$$

The double-exponential selectivity has non-estimable parameters x_1 and x_2, and estimable parameters x_0, y_0, y_1, and y_2. α is a scaling parameter, with default value of $\alpha = 1$. It can be 'U-shaped'. Bounds for x_0 must be such that $x_1 < x_0 < x_2$. With $\alpha = 1$, the selectivity passes through the points (x_1, y_1), (x_0, y_0), and (x_2, y_2). If both y_1 and y_2 are greater than y_0 the selectivity is 'U-shaped' with minimum at (x_0, y_0).

For example, to define an double-exponential selectivity for a model with ages 1 to 6 that passes through the points $(1,1)$, $(3,0.1)$, and $(6,1)$, use

```
@selectivity Sel
type double_exponential
alpha 1
x0 3
y0 0.1
x1 1
y1 1
x2 6
y2 1
```

4.13.11. spline

The spline selectivity implements a cubic spline that has non-estimable knots, and an estimable value for each knot. The cubic spline is either (i) a natural splines where the where the second derivatives are set to 0 at the boundaries, i.e., the values at the boundaries are horizontal, (ii) a spline with a fixed first derivative at the boundaries (linear, but not necessarily horizontal) and (iii) spline which turns into a parabola at the boundaries.

For example, to define a spline selectivity for a model with ages 1 to 6 with 3 knots defined for the ages 1, 3, and 6, and where the shape of the selectivity is horizontal at the ends, use

```
@selectivity Sel
type spline
alpha 1
knots 1 3 6
values 0.1 0.5 1.0
method natural
```

5. The estimation section

5.1. Role of the estimation section

The role of the estimation section is to define the tasks carried out by `SPM`:

1. Define the objective function (see Section 5.2)
2. Define the parameters to be estimated (see Section 5.3)
3. Calculate a point estimate, i.e., the maximum posterior density estimate (MPD) (see Section 5.4).
4. Calculate a posterior profile selected parameters, i.e., find, for each of a series of values of a parameter, allowing the other estimated parameters to vary, the minimum value of the objective function (see Section 5.5).
5. Generate an MCMC sample from the posterior distribution (see Section 5.6).
6. Calculate the approximate covariance matrix of the parameters as the inverse of the minimizer's approximation to the Hessian, and the corresponding correlation matrix (see Section 5.4).

The estimation section defines the objective function, parameters of the model, and the method of estimation (point estimates, Bayesian posteriors, profiles, etc.). The objective function is based on a goodness-of-fit measure of the model to observations, priors and penalties. See the observation section for a description of the observations, likelihoods, priors and penalties.

5.2. The objective function

In Bayesian estimation, the objective function is a negative log-posterior,

$$Objective(p) = -\sum_i \log\left[L\left(\mathbf{p}|O_i\right)\right] - \log\left[\pi\left(\mathbf{p}\right)\right] \tag{5.1}$$

where π is the joint prior density of the parameters p.

The contribution to the objective function from the likelihoods are defined in Section 6.1. In addition to likelihoods, priors (see Section 5.7) and penalties (see Section 5.8) are components of the objective function.

Penalties can be used to ensure that the exploitation rate constraints on mortality events (i.e., fisheries) are not breached (otherwise there is nothing to prevent the model from having abundances so low that the recorded mortalities could not have been taken), penalties on category transitions (to ensure there are enough individuals to move), and possibly penalties to encourage estimated values to be similar or smooth, etc.

5.3. Specifying the parameters to be estimated

The estimable parameters that will be estimated are defined using `@estimate` commands (see Section 9). An `@estimate` command-block looks like,

```
@estimate process[MyRecruitment].r0
```

```
lower_bound 1000
upper_bound 100000
type uniform
```

where `type` specifies the type of prior.

See Section 3.5.5 for instructions on how to generate the parameter name. At least one parameter to be estimated if doing an estimation, profile, or MCMC run. Initial values for the parameters to be estimated will still need to be provided, and these are used as the starting values for the minimiser. However, these may be overwritten if you provide a set of alternative starting values (i.e., using `spm -i`, see Section 3.4).

All parameters are estimated within bounds. For each parameter to be estimated, you need to specify the bounds and the prior (Section 5.7). Note that the bounds and prior for each parameter refer to the values of the parameters, not the actual values resulting from the application of the parameter to an equation. If estimating only some elements of a vector, either define the elements of the vector to be estimated (see 3.5.5) or fix the others by setting the bounds equal.

The estimation of parameters can be phased. Here, some of the estimated parameters are initially held fixed, and a minimisation is carried out. Next, some or all of the remaining parameters that were initially held fixed are freed, and another minimisation is carried out. This process continues until all phases have been carried out.

5.4. Point estimation

Point estimation is invoked with `spm -e`. Mathematically, it is an attempt to find a minimum of the objective function. SPM has two algorithms for solving (minimising) the optimisation problem. The first uses a quasi-Newton minimiser built which is a slightly modified implementation of the main algorithm of Dennis Jr. & Schnabel (Dennis Jr and Schnabel, 1996), while the second uses a genetic algorithm developed by Storn & Price (Storn and Price, 1995), the differential evolution minimiser . However, the second minimiser does not produce an estimate of the covariance matrix, and hence cannot be used to start an MCMC.

5.4.1. The numerical differences minimiser

The minimiser has three kinds of (non-error) exit status:

1. Successful convergence (suggests you have found a local minimum, at least).
2. Convergence failure (you have not reached a local minimum, though you may deem yourself to be ‘close enough’ at your own risk).
3. Convergence unclear (the minimiser halted but was unable to determine if convergence occurred. You may be at a local minimum, although you should check by restarting the minimiser at the final values of the estimated parameters).

You can choose the maximum number of quasi-Newton iterations and objective function evaluations allotted to the minimiser. If it exceeds either limit, it exits with a convergence failure. We recommend large numbers of evaluations and iterations (at least the defaults of 300 and 1000) unless you successfully reach convergence with less. You can also specify an alternative starting point of the minimiser using `spm -i`.

We want to stress that this is a local optimisation algorithm trying to solve a global optimisation problem. What this means is that, even if you get a ’successful convergence’ message, your solution

may be only a local minimum, not a global one. To diagnose this problem, try doing multiple runs from different starting points and comparing the results, or doing profiles of one or more key parameters and seeing if any of the profiled estimates finds a better optimum than than the original point estimate.

The approximate covariance matrix of the estimated parameters can be calculated as the inverse of the minimiser's approximation to the Hessian, and the corresponding correlation matrix is also calculated. Be aware that

- the Hessian approximation develops over many minimiser steps, so if the minimiser has only run for a small number of iterations the covariance matrix can be a very poor approximation
- the inverse Hessian is not a good approximation to the covariance matrix of the estimated parameters, and may not be useful to construct, for example, confidence intervals.

Also note that if an estimated parameter has equal lower and upper bounds, it will have entries of '0' in the covariance matrix and `NaN` or `-1.#IND` (depending on the operating system) in the correlation matrix.

5.4.2. The differential evolution minimiser

The differential evolution minimiser is a simple population based, stochastic function minimizer, but is claimed to be quite powerful in solving minimisation problems. It is a method of mathematical optimization of multidimensional functions and belongs to the class of evolution strategy optimizers. Initially, the procedure randomly generates and evaluates a number of solution vectors (the population size), each with p parameters. Then, for each generation (iteration), the algorithm creates a candidate solution for each existing solution by random mutation and uniform crossover. The random mutation generates a new solution by multiplying the difference between two randomly selected solution vectors by some scale factor, then adding the result to a third vector. Then an element-wise crossover takes place with probability P_{cr}, to generate a potential candidate solution. If this is better than the initial solution vector, it replaces it, otherwise the original solution is retained. The algorithm is terminated after either a predefined number of generations (`max_generations`) or when the maximum difference between the scaled individual parameters from the candidate solutions from all populations is less than some predefined amount `tolerance`.

The differential evolution minimiser can be good at finding global minimums in surfaces that may have local minima. However, the speed of the minimiser, and the ability to find a good minima depend on the number of initial 'populations'. Some authors recommend that the number of populations be set at about $10 * p$, where p is the number of free parameters. However, depending on your problem, you may find that you may need more, or that less will suffice.

We note that there is no proof of convergence for the differential evolution solver, but several papers have found it to be an efficient method of solving multidimensional problems. Our (limited) experience suggests that it can often find a better minima and may be faster or longer (depending on the actual model specification) at finding a solution when compared with the numerical differences minimiser. Comparisons with auto-differentiation minimisers or other more sophisticated algorithms have not been made.

5.5. Posterior profiles

If profiles are requested `spm -p`, `SPM` will first calculate a point estimate. For each scalar parameter or, in the case of vectors or selectivities, the element of the parameter to be profiled, `SPM` will fix its

value at a sequence of n evenly spaced numbers ($step$) between a specified lower and upper bounds l and u, and calculate a point estimate at each value.

By default $step = 10$, and $(l,u) =$ (lower bound on parameter plus $(range/(2n))$, upper bound on parameter less $(range/(2n))$. Each minimisation starts at the final parameter values from the previous resulting value of the parameter being profiled. SPM will report the objective function for each parameter value. Note that an initial point estimate should be compared with the profile, not least to check that none of the other points along the profile have a better objective function value than the initial 'minimum'.

You specify which parameters are to be profiled, and optionally the number of steps, lower bound, and upper bound for each. In the case of vector parameters, you will also need to specify the element of the vector being profiled.

You can also supply the initial starting point for the estimation using `spm -i file` — this may improve the minimiser performance for the profiles.

If you get an implausible profile, it may be a result of not using enough iterations in the minimiser or a poor choice of minimiser control variables (e.g., the minimiser tolerance). It also may be useful to try both if the minimisers in SPM and compare the results.

5.6. Bayesian estimation

SPM can use a Markov Chain Monte Carlo (MCMC) to generate a sample from the posterior distribution of the estimated parameters `spm -m` and output the sampled values to a file (optionally keeping only every nth set of values).

As SPM has no post-processing capabilities. SPM cannot produce MCMC convergence diagnostics (use a package such as BOA) or plot/summarize the posterior distributions of the output quantities (for example, using a general-purpose statistical or spreadsheet package such as S-Plus, **R**, or Microsoft Excel).

Bayesian methodology and MCMC are both large and complex topics, and we do not describe either properly here. See Gelman et al. (1995) and Gilks et al. (1994) for details of both Bayesian analysis and MCMC methods. In addition, see Punt & Hilborn (2001) for an introduction to quantitative fish stock assessment using Bayesian methods.

This section only briefly describes the MCMC algorithms used in SPM. See Section 9.3 for a better description of the sequence of SPM commands used in a full Bayesian analysis.

SPM uses a straightforward implementation of the Metropolis-Hastings algorithm (Gelman et al., 1995, Gilks et al., 1994). The Metropolis-Hastings algorithm attempts to draw a sample from a Bayesian posterior distribution, and calculates the posterior density π, scaled by an unknown constant. The algorithm generates a 'chain' or sequence of values. Typically the beginning of the chain is discarded and every Nth element of the remainder is taken as the posterior sample. The chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

- Draw a candidate step s from a proposal distribution J, which should be symmetric i.e., $J(-s) = J(s)$.
- Calculate $r = min(\pi(x_i + s)/\pi(x_i), 1)$.
- Let $x_i + 1 = x_i + s$ with probability r, or x_i with probability $1 - r$.

An initial point estimate is produced before the chain starts, which is done so as to calculate the

approximate covariance matrix of the estimated parameters (as the inverse Hessian), and may also be used as the starting point of the chain.

The user can specify the starting point of the point estimate minimiser using `spm -i`. Don't start it too close to the actual estimate (either by using `spm -i`, or by changing the initial parameter values in input configuration file) as it takes a few iterations to form a reasonable approximation to the Hessian.

There are two options for the starting point of the Markov Chain:

- Start from the point estimate.
- Start from a random point near the point estimate (the point is generated from a multivariate normal distribution, centred on the point estimate, with covariance equal to the inverse Hessian times a user-specified constant). This may be useful if the chain gets 'stuck' at the point estimate, or if you wish to generate multiple chains from for later MCMC diagnostic tests.

The chain moves in natural space, i.e., no transformations are applied to the estimated parameters. The default proposal distribution is a multivariate t centred on the current point, with covariance matrix equal to a matrix based on the approximate covariance produced by the minimiser, times some stepsize factor. The following steps define the initial covariance matrix of the proposal distribution:

- The covariance matrix is taken as the inverse of the approximate Hessian from the quasi-Newton minimiser.
- The covariance matrix is modified so as to decrease all correlations greater than `@mcmc.max_correlation` down to `@mcmc.max_correlation`, and similarly to increase all correlations less than `-@mcmc.max_correlation` up to `-@mcmc.max_correlation` (the `@mcmc.max_correlation` parameter defaults to 0.8). This should help to avoid getting 'stuck' in a lower-dimensional subspace.
- The covariance matrix is then modified either by,
  - if `@mcmc.adjustment_method=covariance`: that if the variance of the *i*th parameter is non-zero and less than `@mcmc.min_difference` times the difference between the parameters' lower and upper bound, then the variance is changed, without changing the associated correlations, to $k = \text{min\_diff}(upper\_bound_i - lower\_bound_i)$. This is done by setting

    $$\text{Cov}(i,j)' = \text{sqrt}(k)\,\text{Cov}(i,j)/\text{sd}(i)$$

    for $i \neq j$, and $\text{var}(i)' = k$
  - if `@mcmc.adjustment_method=correlation`: that if the variance of the *i*th parameter is non-zero and less than `@mcmc.min_difference` times the difference between the parameters' lower and upper bound, then its variance is changed to $k = min\_diff(upper\_bound_i - lower\_bound_i)$. This differs from (i) above in that the effect of this option is that it also modifies the resulting correlations between the *i*th parameter and all other parameters.

  This allows each estimated parameter to move in the MCMC even if its variance is very small according to the inverse Hessian. In both cases, the `@mcmc.min_difference` parameter defaults to 0.0001.
- The `@mcmc.stepsize` (a scalar factor applied to the covariance matrix to improve the acceptance probability) is chosen by the user. The default is $2.4d^{-0.5}$ where d is the number

of estimated parameters, as recommended by Gelman et al. (Gelman et al., 1995). However, you may find that a smaller value may often be better.

The proposal distribution can also change adaptively during the chain, using two different mechanisms. Both are offered as means of improving the convergence properties of the chain. It is important to note that any adaptive behaviour must finish before the end of the burn-in period, i.e., the proposal distribution must be finalised before the kept portion of the chain starts. The adaptive mechanisms are as follows:

1. You can request that the stepsize change adaptively at one or more sample numbers. At each adaptation, the stepsize is doubled if the acceptance rate since the last adaptation is more than 0.5, or halved if the acceptance rate is less than 0.2. (See Gelman et al. (Gelman et al., 1995) for justification.)

The probability of acceptance for each jump is 0 if it would move out of the bounds, or 1 if it improves the posterior, or (new posterior/old posterior) otherwise. You can specify how often the position of the chain is recorded using the keep parameter. For example, with keep 10, only every 10th sample is recorded.

You have the option to specify that some of the estimated parameters are fixed during the MCMC. If the chain starts at the point estimate or at a random location, these fixed parameters are set to their values at the point estimate.

The posterior sample can also be used for simulations (Section sec:simulation-observations) with the values supplied using `spm -i file`.

5.7. Priors

In a Bayesian analysis, you need to give a prior for every parameter that is being estimated. There are no default priors.

Note that when some of these priors are parameterised in terms of mean, c.v., and standard deviation, these refer to the parameters of the distribution before bounds are applied. The moments of the prior after the bounds are applied may differ.

`SPM` has the following priors (expressed in terms of their contribution to the objective function):

1. Uniform

$$-\log(\pi(p)) = 0 \tag{5.2}$$

2. Uniform-log (i.e., $\log(p) \sim$ uniform)

$$-\log(\pi(p)) = \log(p) \tag{5.3}$$

3. Normal with mean μ and c.v. c

$$-\log(\pi(p)) = 0.5\left(\frac{p-\mu}{c\mu}\right)^2 \tag{5.4}$$

4. Normal with mean μ and standard deviation σ

$$-\log(\pi(p)) = 0.5\left(\frac{p-\mu}{\sigma}\right)^2 \tag{5.5}$$

5. Lognormal with mean μ and c.v. c

$$-\log(\pi(p)) = \log(p) + 0.5\left(\frac{\log(p/\mu)}{s} + \frac{s}{2}\right)^2 \tag{5.6}$$

where s is the standard deviation of $\log(p)$ and $s = \sqrt{\log(1+c^2)}$.

6. Beta with mean μ and standard deviation σ, and range parameters A and B

$$-\log(\pi(p)) = (1-m)\log(p-A) + (1-n)\log(B-p) \tag{5.7}$$

where $\nu = \frac{\mu-A}{B-A}$, and $\tau = \frac{(\mu-A)(B-\mu)}{\sigma^2} - 1$ and then $\mu = \tau\nu$ and $n = \tau(1-\nu)$. Note that the beta prior is undefined when $\tau \leq 0$.

5.8. Penalties

Penalties can be used to encourage or discourage parameter values or model outputs that are unlikely to be sensible, by adding a penalty to the objective function. For example, parameter estimates that do not allow a known mortality event to remove enough individuals from the population can be discouraged with an event mortality penalty. `SPM` requires penalty functions for processes that move or shift a *number* of individuals between categories or from the partition.

For most penalties, you need to specify a multiplier, and the objective function is increased by this multiplier times the penalty value as described below. In some cases you will need to make the multiplier quite large to prohibit some model behaviour.

Currently, the penalties for the processes `@process[label].type=event_mortality` and `@process[label].type=category_transition` are the only penalties implemented.

For both of these processes, two types of penalty can be defined, natural scale (the default) and log scale. Both of these types add a penalty value of the squared difference between the observed value (i.e., the actual number of individuals to be removed in an event mortality process or the actual number of individuals to shift in a category transition process), and the number that were moved (if less than or equal), times the penalty multiplier.

The natural scale penalty just uses at the squared difference on a natural scale, while the log scale penalty uses the squared difference of the logged values.

6. The observation section

6.1. Observations and likelihoods

Observations are typically supplied as observations at an instance in time, over some spatially aggregated area. Time series of observations can be supplied as separate observations for each year or point in time.

`SPM` allows the following types of observations;

- Observations of proportions by age class within categories
- Observations of proportions between categories within age classes
- Relative and absolute abundance/biomass observations
- Relative proportions present observations

The definitions for each type of observation are described below, including how the observed values should be supplied, how `SPM` calculates the expected values, and the likelihoods that are available for each type of observation.

`SPM` evaluates the observations at the end of a time-step (i.e., after all of the processes for that time-step have been applied). However, the observation can be applied to the abundance at the start of a time-step or part-way through a time-step by the use of the `proportion_time_step` subcommand.

By default (i.e., if `proportion_method = mean`), the partition at some point p during the time-step is then evaluated as the weighted sum between the start and end of the time-step, i.e, for any element i in the partition, $n_i = (1-p)n_i^{start} + pn_i^{end}$. Note that it may not be sensible to use a value other than one, depending on the processes that happen during the time-step (for example, if the time-step contains an ageing process).

If the `proportion_method = difference`, then the observation is of the *difference* between the population state at the start of the time-step and the end. This can be used to generate expected values for observations of, for example removals due to a mortality event, by only having a single process in the time-step. In this case, the proportion_time_step is simply a multiplier of the population state.

6.2. Proportions-at-age observations

Proportions-at-age observations are observations of either the relative number of individuals at age or relative biomass at age, via some selectivity.

The observation is supplied for a given year and time-step, for some selected age classes of the population (i.e., for a range of ages multiplied by a selectivity), for categories aggregated over a set of spatial cells. Note that the categories defined in the observations must have an associated selectivity, defined by `selectivities`.

The age range must be ages defined in the partition (i.e., between `@model.min_age` and `@model.max_age` inclusive), but the upper end of the age range can optionally be a plus group — which must be either the same or less than the plus group defined for the partition.

Proportions-at-age observations can be supplied as;

1. a set of proportions for a single category,
2. a set of proportions for multiple categories, or
3. a set of proportions across aggregated categories.

For example, for a model with the two categories *male* and *female*, we might supply either (i) a set of proportions for a single category (i.e., males) within each age class; (ii) a set of proportions describing the proportions of individuals within each age class across multiple categories (i.e., males and females) simultaneously, or (iii) a set of proportions for the total number of individuals over the aggregated categories (i.e., males + females) combined, within each age class.

The way the categories of the observation are defined specifies which of these alternatives are used. It is also possible to have an observation with multiple and aggregated categories simultaneously.

Proportions-at-age for a single category

This form of defining the observation is the simplest, and is used to model a set of proportions of a single category by age class. For example, to specify that the observations are of the proportions of male within each age class, then the subcommand `categories` for the `@observation[label].type=proportion_by_age` command is,

```
categories male
```

`SPM` then expects that there will be a single vector of proportions supplied, with one proportion for each age class within the defined age range, and that these proportions sum to one.

For example, if the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10). The expected values will be the expected proportions of males within each of these age classes (after ignoring any males aged less than 3 or older than 10), after applying a selectivity at the year and time-step specified. The supplied vector of proportions (i.e., in this example, the 8 proportions) must sum to one, which is evaluated with a default tolerance of 0.001.

The observations must be also supplied using all or some of the the values of defined by some *categorical* layer. `SPM` calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

For example, in a 2×2 spatial model a categorical layer (e.g., with label Area) may define that cells $(1,1)$ and $(1,2)$ have value A and cells $(2,1)$ and $(2,2)$ have value B, i.e.,

```
@layer Area
type categorical
data A A
data B B
```

The observations for those spatial cells where the categorical layer has value A would be,

```
@observation MyProportions
type proportions_at_age
layer Area
...
categories male
min_age 1
max_age 5
obs A 0.01 0.09 0.20 0.30 0.40
...
```

Or, for both *A* and *B* as,

```
@observation MyProportions
type proportions_at_age
layer Area
...
categories male
min_age 1
max_age 5
obs A 0.01 0.09 0.20 0.30 0.40
obs B 0.02 0.06 0.12 0.25 0.55
...
```

Note that to have an observation for each individual spatial cell in a model, then define a categorical layer that has a single, unique value for each spatial cell for use in the observation.

Proportions-at-age for multiple categories

This form of the observation extends the idea above for multiple categories. It is used to model a set of proportions over several categories by age class. For example, to specify that the observations are of the proportions of male or females within each age class, then the subcommand `categories` for the `@observation[label].type=proportion_by_age` command is,

```
categories male female
```

`SPM` then expects that there will be a single vector of proportions supplied, with one proportion for each category and age class combination, and that these proportions sum to one.

For example, if there were two categories and the age range was 3 to 10, then 16 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10, for each category male and female). The expected values will be the expected proportions of males and within each of these age classes (after ignoring those aged less than 3 or older than 10), after applying a selectivity at the year and time-step specified. The supplied vector of proportions (i.e., in this example, the 16 proportions) must sum to one, which is evaluated with a default tolerance of 0.001.

For example, using the earlier spatial model with a categorical layer that has label Area, the observations for those spatial cells where the categorical layer has value *A* would be,

```
@observation MyProportions
type proportions_at_age
layer Area
...
categories male female
min_age 1
max_age 5
obs A 0.01 0.05 0.10 0.20 0.20 0.01 0.05 0.15 0.20 0.03
obs B 0.02 0.06 0.10 0.21 0.18 0.02 0.05 0.15 0.20 0.01
...
```

Proportions-at-age across aggregated categories

This form of the observation extends the idea above, but allows categories to be aggregated before the proportions are calculated. It is used to model a set of proportions from several categories that have been combined by age class. To indicate that two (or more) categories are to be aggregated,

separate them with a '+' symbol. For example, to specify that the observations are of the proportions of male and females combined within each age class, then the subcommand `categories` for the `@observation[label].type=proportion_by_age` command is,

```
categories male + female
```

SPM then expects that there will be a single vector of proportions supplied, with one proportion for each age class, and that these proportions sum to one.

For example, if there were two categories and the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10, for the sum of males and females within each age class). The expected values will be the expected proportions of males + females within each of these age classes (after ignoring those aged less than 3 or older than 10), after applying a selectivity at the year and time-step specified. The supplied vector of proportions (i.e., in this example, the 16 proportions) must sum to one, which is evaluated with a default tolerance of 0.001.

For example, using the earlier spatial model with a categorical layer that has label Area, the observations for those spatial cells where the categorical layer has value *A* would be,

```
@observation MyProportions
type proportions_at_age
layer Area
...
categories male + female
min_age 1
max_age 5
obs A 0.02 0.13 0.25 0.30 0.30
obs B 0.02 0.06 0.18 0.35 0.39
...
```

The later form can then be extended to include multiple categories, or multiple aggregated categories. For example, to describe proportions for the three groups: immature males, mature males, and all females (immature and mature females added together) for ages 1–4, a total of 12 proportions are required

```
@observation MyProportions
type proportions_at_age
layer Area
...
categories male_immature male_mature female_immature + female_mature
min_age 1
max_age 4
obs A 0.05 0.15 0.15 0.05 0.02 0.03 0.08 0.04 0.05 0.15 0.15 0.08
...
```

6.2.1. Likelihoods for proportions-at-age observations

SPM implements three likelihoods for proportions-at-age observations, the Dirichlet distribution, the multinomial likelihood, and the lognormal likelihood.

The Dirichlet likelihood

For the observed proportions at age O_i for age classes i, with sample size N, and the expected proportions at the same age classes E_i, the negative log-likelihood is defined as;

$$-\log(L) = -\log(\Gamma\sum_i(\alpha_i)) + \sum_i \log(\Gamma(\alpha_i)) - \sum_i(\alpha_i - 1)\log(Z(O_i,\delta)) \tag{6.1}$$

where $\alpha_i = Z(NE_i, \delta)$, $\sum_i O_i = 1$, and $\sum_i E_i = 1$. $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta, \delta)$ is defined as,

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.2}$$

The default value of δ is 1×10^{-11}.

The multinomial likelihood

For the observed proportions at age O_i for age classes i, with sample size N, and the expected proportions at the same age classes E_i, the negative log-likelihood is defined as;

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(Z(E_i, \delta)) \tag{6.3}$$

where $\sum_i O_i = 1$ and $\sum_i E_i = 1$. $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta, \delta)$ is defined as,

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.4}$$

The default value of δ is 1×10^{-11}.

The lognormal likelihood

For the observed proportions at age O_i for age classes i, with c.v. c_i, and the expected proportions at the same age classes E_i, the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i / Z(E_i, \delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{6.5}$$

where

$$\sigma_i = \sqrt{\log\left(1 + c_i^2\right)} \tag{6.6}$$

and the c_i's are the c.v.s for each age class i, and $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta, \delta)$ is defined as,

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.7}$$

The default value of δ is 1×10^{-11}.

6.3. Proportions-by-category observations

Proportions-by-category observations are observations of either the relative number of individuals between categories within age classes, or relative biomass between categories within age classes.

The observation is supplied for a given year and time-step, for some selected age classes of the population (i.e., for a range of ages multiplied by a selectivity), for categories aggregated over a set of spatial cells.

The age range must be ages defined in the partition (i.e., between `@model.min_age` and `@model.max_age` inclusive), but the upper end of the age range can optionally be a plus group — which may or may not be the same as the plus group defined for the partition.

Proportions-by-category observations can be supplied for any set of categories as a proportion of themselves and any set of additional categories. For example, for a model with the two categories *male* and *female*, we might supply observations of the proportions of males in the population at each age class. The subcommand `categories` defines the categories for the numerator in the calculation of the proportion, and the subcommand `categories2` supplies the additional categories to be used in the denominator of the calculation. In addition, each category must have an associated selectivity, defined by `selectivities` for the numerator categories and `selectivities2` for the additional categories used in the denominator, e.g.,

```
categories male
categories2 female
selectivities male-selectivity
selectivities2 female-selectivity
```

defines that the proportion of males in each age class as a proportion of males + females. `SPM` then expects that there will be a vector of proportions supplied, with one proportion for each age class within the defined age range, i.e., if the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10). The expected values will be the expected proportions of male to male + female within each of these age classes, after applying the selectivities at the year and time-step specified.

The observations must be supplied using all or some of the values defined by a categorical layer. `SPM` calculates the expected values by summing over the ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

For example, in a 2×2 spatial model a categorical layer (e.g., with label Area) may define that cells $(1,1)$ and $(1,2)$ have value *A* and cells $(2,1)$ and $(2,2)$ have value *B*, i.e.,

```
@layer Area
type categorical
data A A
data B B
```

Here we supply observations for those spatial cells where the categorical layer has value *A* as,

```
@observation MyProportions
type proportions_by_category
layer Area
...
categories male
categories2 female
min_age 1
max_age 5
obs A 0.01 0.05 0.10 0.20 0.20
...
```

Or, for both *A* and *B* as,

```
@observation MyProportions
type proportions_by_category
layer Area
...
categories male
categories2 female
min_age 1
max_age 5
obs A 0.01 0.05 0.10 0.20 0.20
obs B 0.02 0.06 0.10 0.21 0.18
...
```

To supply an observation for individual spatial cells, then you will need to define a categorical layer with a single, unique value for each spatial cell.

6.3.1. Likelihoods for proportions-by-category observations

SPM implements two likelihoods for proportions-by-category observations, the binomial likelihood, and the normal approximation to the binomial (binomial-approx).

The binomial likelihood

For observed proportions O_i for age class i, where E_i are the expected proportions for age class i, and N_i is the effective sample size for age class i, then the negative log-likelihood is defined as;

$$\begin{aligned} -\log(L) = -\sum_i [\, &\log(N_i!) - \log((N_i(1-O_i))!) - \log((N_iO_i)!) + N_iO_i\log(Z(E_i,\delta)) \\ &+ N_i(1-O_i)\log(Z(1-E_i,\delta))\,] \end{aligned} \tag{6.8}$$

where $Z(\theta,\delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta,\delta)$ is defined as,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.9}$$

The default value of δ is 1×10^{-11}.

The normal approximation to the binomial likelihood

For observed proportions O_i for age class i, where E_i are the expected proportions for age class i, and N_i is the effective sample size for age class i, then the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{6.10}$$

where $Z(\theta,\delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta,\delta)$ is defined as,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.11}$$

The default value of δ is 1×10^{-11}.

6.4. Abundance or biomass observations

Abundance (or biomass) observations are observations of either a relative or absolute number (or biomass) of individuals from a set of categories after applying a selectivity. The observations classes are the same, except that a biomass observation will use the biomass as the observed (and expected) value (calculated from mean weight of individuals within each age and category) while an abundance observation is just the number of individuals.

Each observation is for a given year and time-step, for some selected age classes of the population (i.e., for a range of ages multiplied by a selectivity), for categories aggregated over a set of spatial cells. Further, you need to provide the label of the catchability coefficient q, which can either be estimated or fixed. For absolute abundance or absolute biomass observations, define a catchability where $q = 1$.

The observations can be supplied for any set of categories. For example, for a model with the two categories *male* and *female*, we might supply an observation of the total abundance/biomass (male + female) or just male abundance/biomass. The subcommand `categories` defines the categories used to aggregate the abundance/biomass. In addition, each category must have an associated selectivity, defined by `selectivities`. For example,

```
categories male
selectivities male-selectivity
```

defines an observation for males after applying the selectivity male-selectivity. `SPM` then expects that there will be a single observation supplied. The expected values for the observations will be the expected abundance (or biomass) of males, after applying the selectivities, at the year and time-step specified.

The observations must be supplied using all or some of the the values of defined by a categorical layer. `SPM` calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

For example, in a 2×2 spatial model a categorical layer (e.g., with label Area) may define that cells $(1,1)$ and $(1,2)$ have value A and cells $(2,1)$ and $(2,2)$ have value B, i.e.,

```
@layer Area
type categorical
data A A
data B B
```

Here we supply abundance observations for those spatial cells where the categorical layer has value A as,

```
@observation MyAbundance
type abundance
layer Area
...
categories male
obs A 1000
...
```

Or, for both A and B as,

```
@observation MyAbundance
type abundance
```

```
layer Area
...
categories male
obs A 1000
obs B 1200
...
```

To supply an observation for individual spatial cells, then you will need to define a categorical layer with a single, unique value for each spatial cell.

Note that, to define a biomass observation instead of an abundance observation, use

```
@observation MyBiomass
type biomass
...
```

6.4.1. Likelihoods for abundance observations

The lognormal likelihood

For observations O_i, c.v. c_i, and expected values qE_i, the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i/qZ(E_i,\delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{6.12}$$

where

$$\sigma_i = \sqrt{\log\left(1+c_i^2\right)} \tag{6.13}$$

and $Z(\theta,\delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta,\delta)$ is defined as,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.14}$$

The default value of δ is 1×10^{-11}.

The normal likelihood

For observations O_i, c.v. c_i, and expected values qE_i, the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \left(\log(c_i E_i) + 0.5 \left(\frac{O_i - E_i}{Z(c_i E_i, \delta)} \right)^2 \right) \tag{6.15}$$

and $Z(\theta,\delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta,\delta)$ is defined as,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.16}$$

The default value of δ is 1×10^{-11}.

6.5. Relative proportions present observations

Relative proportions present observations are observations of a relative proportion of sampling efforts that detected the presence of one of more individuals from a set of categories after applying a selectivity. This assumes that the probability of detecting an individual in a cell is proportional to the density (in number of individuals per unit area) for that cell, i.e., the expected proportion present E_i is;

$$E_i = \begin{cases} 1, & \text{where } qN_i/A_i \geq 1 \\ qN_i/A_i, & \text{otherwise} \end{cases} \tag{6.17}$$

where q is the catchability coefficient, N_i is the number of individuals in cell i, and A_i is the area of cell i as given in the base layer.

Each observation is for a given year and time-step, for some selected age classes of the population (i.e., for a range of ages multiplied by a selectivity), for categories aggregated over a set of spatial cells. Further, you need to provide the label of the catchability coefficient q, which can either be estimated or fixed.

The observations can be supplied for any set of categories. For example, for a model with the two categories *male* and *female*, we might supply an observation of the total presence proportions (male + female) or just male proportions. The subcommand `categories` defines the categories used to aggregate. In addition, each category must have an associated selectivity, defined by `selectivities`. For example,

```
categories male
selectivities male-selectivity
```

defines an observation for males after applying the selectivity male-selectivity. `SPM` then expects that there will be a single observation supplied. The expected values for the observations will be the expected relative probability of presence for males, after applying the selectivities, at the year and time-step specified.

The observations must be supplied using all or some of the the values of defined by a categorical layer. `SPM` calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories for those cells where the categorical layer has the same value as defined for each vector of observations.

For example, in a 2×2 spatial model a categorical layer (e.g., with label Area) may define that cells $(1,1)$ and $(1,2)$ have value A and cells $(2,1)$ and $(2,2)$ have value B, i.e.,

```
@layer Area
type categorical
data A A
data B B
```

Here we supply presence observations for those spatial cells where the categorical layer has value A as,

```
@observation MyPresence
type presence
layer Area
...
categories male
obs A 0.10
...
```

Or, for both A and B as,

```
@observation MyPresence
type presence
layer Area
...
categories male
obs A 0.10
obs B 0.12
...
```

To supply an observation for individual spatial cells, then you will need to define a categorical layer with a single, unique value for each spatial cell.

6.5.1. Likelihoods for presence observations

The binomial likelihood

For observed proportions O_i for cell i, we define E_i as the expected proportion in cell i, where $E_i = \min(qD_i, 1)$ and d_i is the density in cell i and q is a catchability coefficient such that $q > 0$. Then if N_i is the effective sample size in cell i, the negative log-likelihood is defined as;

$$\begin{aligned} -\log(L) = -\sum_i [\, &\log(N_i!) - \log((N_i(1-O_i))!) - \log((N_i O_i)!) + N_i O_i \log(Z(qE_i, \delta)) \\ &+ N_i(1-O_i)\log(Z(1-qE_i, \delta))\,] \end{aligned} \tag{6.18}$$

where $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta, \delta)$ is defined as,

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.19}$$

The default value of δ is 1×10^{-11}.

6.6. Process error

Additional 'process error' can be defined for each set of observations. Additional process error has the effect of increasing the observation error in the data, and hence of decreasing the relative weight given to the data in the fitting process.

For observations where where the likelihood is parameterised by the c.v., you can specify the process error for a given set of observations as a c.v., in which case all the c.v.s c_i are changed to

$$c_i' = \sqrt{c_i^2 + c_{process\_error}^2} \tag{6.20}$$

Note that $c_{process\_error} \geq 0$, and that $c_{process\_error} = 0$ is equivalent to no process error.

For the multinomial and binomial likelihoods that are parameterised by the effective sample size N, then,

$$N_i' = \frac{1}{1/N_i + 1/N_{process\_error}} \tag{6.21}$$

Note that this requires that $N_{process\_error} > 0$, but we allow the special case of $N_{process\_error} = 0$, and define $N_{process\_error} = 0$ as no process error (i.e., defined to be equivalent to $N_{process\_error} = \infty$).

For the Dirichlet likelihood (parameterised by the effective sample size N) then,

$$N'_i = N_i N_{process\_error} \tag{6.22}$$

Note that this requires that $0 < N_{process\_error} \geq 1$. As above, we allow the special case of $N_{process\_error} = 0$ and define this as no process error (i.e., defined to be equivalent to $N_{process\_error} = 1.0$) for consistency with other likelihoods.

For both the c.v. and N process errors, the process error has more effect on small errors than on large ones. Note that a large *value* for the N process error in the case of the multinomial and binomial means a *small* process error.

6.7. Ageing error

`SPM` can apply ageing error age frequency observations. Ageing error is applied to the expected values for proportions-at-age observations. The ageing error is applied as a misclassification matrix, which has the effect of 'smearing' the age frequencies. These are used in calculating the fits to the observed values, and hence the contribution to the total objective function.

Ageing error is optional, and if it is used, it may be omitted for any individual time series. Different ageing error models may be applied for different observation commands. See Section 7.15 for reporting the misclassification matrix.

The ageing error models implemented are,

1. None: The default model is to apply no ageing error.
2. Off by one: Proportion p_1 of individuals of each age a are misclassified as age $a-1$ and proportion p_2 are misclassified as age $a+1$. Individuals of age $a < k$ are not misclassified. If there is no plus group in the population model, then proportion p_2 of the oldest age class will 'fall off the edge' and disappear.
3. Normal: Individuals of age a are classified as ages which are normally distributed with mean a and constant c.v. c. As above, if there is no plus group in the population model, some individuals of the older age classes may disappear. If c is high enough, some of the younger age classes may 'fall off the other edge'. Individuals of age $a < k$ are not misclassified.

Note that the expected values (fits) reported by `SPM` for observations with ageing error will have had the ageing error applied.

6.8. Simulating observations

`SPM` can generate simulated observations for a given model with given parameter values (using `spm -s`). Simulated observations are randomly distributed values, generated according to the error assumptions defined for each observation, around fits calculated from one or more sets of the 'true' parameter values. Simulating from a set of parameters can be used to generate observations from an operating model or as a form of parametric bootstrap.

The procedure `SPM` uses for simulating observations is to first run using the 'true' parameter values and generate the expected values. Then, if a set of observations uses ageing error, ageing error is applied. Finally a random value for each observed value is generated based on (i) the expected values, (ii) the type of likelihood specified, and (iii) the variability parameters (e.g., `error_value` and `process_error`).

Methods for generating the random error, and hence simulated values, depend on the specific likelihood type of each observation.

1. Normal likelihood parameterised by c.v.: Let E_i be the fitted value for observation i, and c_i be the

corresponding c.v. (adjusted by the process error if applicable). Each simulated observation value S_i is generated as an independent normal deviate with mean E_i and standard deviation $E_i c_i$.

2. Log-normal likelihood: Let E_i be the fitted value for observation i and c_i be the corresponding c.v. (adjusted by the process error if applicable). Each simulated observation value S_i is generated as an independent lognormal deviate with mean and standard deviation (on the natural scale, not the log-scale) of E_i and $E_i c_i$ respectively. The robustification parameter δ is ignored.
3. Dirichlet likelihood: Let E_i be the fitted value for observation i, for i between 1 and n, and let N be the sample size (adjusted by process error if applicable, and then rounded up to the next whole number). The robustification parameter δ is ignored. Then,
   (a) Each simulated observation S_i is generated using the gamma distribution, using the shape parameters $N * E_i$
   (b) The simulated observation values are then rescaled so that their sum is equal to 1
4. Multinomial likelihood: Let E_i be the fitted value for observation i, for i between 1 and n, and let N be the sample size (adjusted by process error if applicable, and then rounded up to the next whole number). The robustification parameter δ is ignored. Then,
   (a) A sample of N values from 1 to n is generated using the binomial distribution, using sample probabilities proportional to the values of E_i
   (b) Each simulated observation value S_i is calculated as the proportion of the N sampled values equalling i
   (c) The simulated observation values S_i are then rescaled so that their sum is equal to 1
5. Binomial and the normal approximation to the binomial likelihoods: Let E_i be the fitted value for observation i, for i between 1 and n, and N_i the corresponding equivalent sample size (adjusted by process error if applicable, and then rounded up to the next whole number). The robustification parameter δ is ignored. Then,
   (a) A sample of N_i independent binary variates is generated, equalling 1 with probability E_i
   (b) The simulated observation value S_i is calculated as the sum of these binary variates divided by N_i

Note that `SPM` will report simulated observations using the usual observation report (`@report[label].type=observation`). The report `@report[label].type=simulated_observation` will generate simulated observations in a form suitable for use as input within a `SPM` input configuration file. See Section 7 for more detail.

6.9. Pseudo-observations

`SPM` can generate expected values for observations without them contributing to the total objective function. These are called pseudo-observations, and can be used to either generate the expected values from `SPM` for reporting or diagnostic purposes. To define an observation as a pseudo-observation, use the command `@observation[label].likelihood=none`. Any observation type can be used as a pseudo-observation. `SPM` can also generate simulated observations from pseudo-observations. Note that;

- Output will only be generated if a report command `@report[label].type=observation` is specified.
- The observed values should be supplied (even if they are 'dummy' observation). These will be processed by `SPM` as if they were actual observation values, and must conform to the validations carried out for the other types of likelihood.
- The subcommands `likelihood`, `obs`, `error_value` and `process_error` have no effect when generating the expected values for the pseudo-observation.
- When simulating observations, `SPM` needs the subcommand `simulation_likelihood` to tell it what sort of likelihood to use. In this case, the `obs`, `error_value` and `process_error` are used to determine the appropriate terms to use for the likelihood when simulating.

7. The report section

The report section specifies the printouts and other outputs from the model. SPM does not, in general, produce any output unless requested by a valid report.

Reports from SPM can be defined to print the spatial structure, partition and states objects at a particular point in time, as well as layers, observation summaries, estimated parameters and objective function values. See below for a more extensive list.

Reports from SPM all conform to a standard style (with one exception — the estimate_values report, see below). The standard style is that reports are prefixed with a user-defined label in square brackets (e.g., [...]), with the second line indicating the type of report, and the report ending with the line *end. For example,

```
[My-report]
report.type: ...
...
*end
```

This syntax should make it easier for external packages to be configured to read SPM output. The extract functions in the **R** spm package uses this information to identify and read SPM output.

Note that the estimate_values report does not print either a header (e.g., [...] or *end at the end of the report. This is as the estimate_values report is designed to provide a single line (or multi-line for more than one set) vector of the estimated parameter values, suitable for reading by SPM (with the command spm -i).

Reports, by default, are directed to standard out (see Section 3.3), but this default can be overwritten be specifying an output file (@report[label].file_name). Hence reports can be directed to separate files as required. Note that if an output file is defined, any data within it is deleted and replaced with the output from the report. To append data to an output file rather than replacing it, use the subcommand @report[label].overwrite=false.

However, some reports *require* that the results are directed to a file (for example, the @report[label].MCMC_samples and @report[label].MCMC_objectives reports). In these cases, a file name must be provided.

Note that reports can be defined that may not be generated. For example printing the partition for a year and/or time-step that does not exist or reporting the covariance matrix when not estimating. Such reports are ignored by SPM and the program will not generate any output for these reports — although they must still conform to SPMs syntax requirements.

Not all reports will be generated in all run modes. Some reports are only available in some run modes. For example, when simulating, only simulation reports will be output.

7.1. Print the model spatial map

Print the spatial co-ordinates of each spatial cell (i.e., row and column labels of each spatial cell) of the spatial structure.

7.2. Print the partition

Print the partition for a given year or given years and time-step. This prints out, for each row and column defined as a valid cell in the base layer (see Section 4.2), the numbers of individuals in each age class in the partition for each year. Note that this report is evaluated at the end of the time-step in the given year(s).

7.3. Print the partition biomass

Print the biomass in the partition for a given year or given years and time-step. This prints out, for each row and column defined as a valid cell in the base layer (see Section 4.2), the biomass of the individuals in each age class in the partition for each year. Note that this report is evaluated at the end of the time-step in the given year(s).

7.4. Print the partition at the end of an initialisation

Print the partition following an initialisation phase. This prints out, for each row and column defined as a valid cell in the base layer (see Section 4.2), the numbers of individuals in each age class in the partition at then end of that initialisation phase.

7.5. Print a process summary

Print a summary of a process. Depending on the process, different summaries are produced. These typically detail the type of process, its parameters and other options, and any associated details.

7.6. Print a preference function summary

Print a summary of a preference function. Depending on the preference function, different summaries are produced. These typically detail the type of preference function, its parameters and other options, and any associated details.

7.7. Print derived quantities

Print out the description of the derived parameter, and the values of the derived quantity as recorded in the model state, for each year of the model. and for all years in the initialisation phases.

7.8. Print derived quantities by cell

Print out the description of the derived quantity by cell, and the values of the derived quantity by cell as recorded in the model state, for each year of the model and optionally for all years in the initialisation phases.

7.9. Print the estimated parameters

Print a summary of the estimated parameters, including the parameter name, lower and upper bounds, the label of the prior, and its value.

7.10. Print the estimated parameters in a vector format

Print the estimated parameter values out as a vector, in a format suitable for use with `spm -i`. The `estimate_values` report prints two lines — a line for the labels of the estimated parameters, and then a line of the values of the estimated parameters. For run modes that produce multi-line output (for example, MCMCs or profiles), only the first line contains the labels of the estimated parameters. All subsequent lines are the values of the estimated parameters only (with each line representing a single set of parameter values).

Note that unlike other reports, the `estimate_values` report does not print either a header (e.g., `[...]` or `*end` at the end.

7.11. Print the objective function

Print the total objective function value, and the value of all observations, the values of all priors, and the value of any penalties that have been incurred in the model. Note that if an individual model run does not incur a penalty, then the penalty will not be reported.

7.12. Print the covariance matrix

Print the Hessian and covariance matrices if estimating and if the covariance has been requested by`@minimiser[label].covariance=true`.

7.13. Print observations, fits, and residuals

Prints out the area (from the observation categorical layer), observed values (as supplied in the input configuration file), expected values as calculated by the model, residuals (observed − expected), the error value, process error, and the total error (i.e., the error value as modified by any additional process error), and the contribution to the total objective function of that individual point in the observation.

Note that constants in likelihoods are often ignored in the objective function score of individual points. Hence, the total score from an observation equals the contribution of the objective function scores from each individual point plus a constant term (if applicable). In likelihoods without a constant term, then the total score from an observation will equal the contribution of the objective function scores from each individual point.

If simulating, then the contribution to the objective function of each observation is reported as zero.

7.14. Print simulated observations

Prints out a complete observation definition (i.e., in the form defined by `@report[label].type=observation`), but with observed values replaced by randomly generated simulated values. The output is in a form suitable for use within a `SPM` input configuration file, reproducing the command and subcommands from the input configuration file.

7.15. Print the ageing error misclassification matrix

Prints out the ageing error misclassification matrix.

7.16. Print layers and meta-layers

Prints the values in the layer (including user supplied layers, abundance and biomass layers, distance and haversine layers, derived quantity layers, and meta-layers) for given year and at the end of a given time-step. Note that the report format is a matrix of values with a value for each spatial cell, except in the case of the distance and haversine layers. For the later, `SPM` reports the distance between each potential source and destination cell as a list.

7.17. Print a derived view via a categorical layer

Prints a summary of the partition, as seen via a categorical layer. Here, values within the spatial cells of a partition are aggregated within the regions defined by the categorical layer for a given age range and given model categories.

7.18. Print selectivities

Prints the values of a selectivity for each age in the partition, for a given year and at then end of a given time-step.

7.19. Print the random number seed

Prints the random number seed used by `SPM` to generate the random number sequence. Future runs made with the same random number seed and the same model will produce identical outputs.

7.20. Print the size-weight relationship

Prints a summary of the size-weight relationship used by `SPM` defined for an age-size relationship by printing the mean weight for a list of sizes. Useful for verifying that the choice of parameters (and their magnitude) are sensible choices.

7.21. Print the size-at-age relationship

Prints a summary of the size-at-age relationship used by `SPM` for each category. For each age between `min_age` and `max_age`, prints the mean size. Useful for verifying that the choice of parameters (and their magnitude) are sensible choices.

7.22. Print the weight-at-age relationship

Prints a summary of the mean weight-at-age relationship used by `SPM` for each category. For each age between `min_age` and `max_age`, prints the mean weight. Useful for verifying that the choice of parameters (and their magnitude) are sensible choices.

7.23. Print the results of an MCMC

Print the MCMC samples, objective function values, and proposal covariance matrix following an MCMC.

7.24. Print the MCMC samples as they are calculated

Print the MCMC samples for each new *i*th sample as they are calculated while doing an MCMC. The output file will be updated with each new sample as it is calculated by `SPM`.

7.25. Print the MCMC objective function values as they are calculated

Print the MCMC objective function values (along with the proposal covariance matrix) for each new *i*th sample as they are calculated while doing an MCMC. The output file will be updated with each new set of objective function values as it is calculated by `SPM`.

8. Population command and subcommand syntax

8.1. Model structure

`@model` Define the spatial structure, population structure, annual cycle, and model years

`nrows` The number of rows n_{rows} in the spatial structure
Type: Integer
Default: No default
Value: A positive integer, $n_{rows} > 0$

`ncols` The number of columns n_{cols} in the spatial structure
Type: Integer
Default: No default
Value: A positive integer, $n_{cols} > 0$

`layer` The label for the base layer
Type: String
Default: No default
Value: Must be a label of a `numeric` layer defined by `@layer`

`cell_length` The length (distance) of one side of a cell
Type: Constant
Default: 1
Value: A positive real number

`categories` Labels of the categories (rows) of the population component of the partition
Type: Vector of strings, of length $1 \ldots n_{categories}$
Default: No default
Value: Names of categories must be unique

`min_age` Minimum age of the population
Type: Integer
Default: No default
Value: A non-negative integer, $age_{min} \geq 0$ and $age_{min} \leq age_{max}$

`max_age` Maximum age of the population
Type: Integer
Default: No default
Value: A non-negative integer, $age_{max} \geq 0$ and $age_{min} \geq age_{min}$

`age_plus_group` Define the largest age as a plus group
Type: Switch
Default: True
Value: Defines the largest age as a plus group

`age_size` Define the label of the associated age-size relationship for each category

Type: Vector of strings, of length $n_{categories}$
Default: No default
Value: Must be labels of command `@age_size`

`initialisation_phases` Define the labels of the phases of the initialisation
Type: Vector of strings, of length of the number of initialisation phases
Default: No default
Value: A valid label defined by `@initialisation_phase`

`initial_year` Define the first year of the model, immediately following initialisation
Type: Integer
Default: No default
Value: Defines the first year of the model, ≥ 1, e.g. 1990

`current_year` Define the current year of the model
Type: Integer
Default: No default
Value: Defines the current year of the model, i.e., the model is run from `@model.first_year` to `@model.current_year`

`time_steps` Define the `@time_step` labels (in order that they are applied) to form the annual cycle
Type: String vector
Default: No default
Value: A valid time_step label, from one of `@time_step`

8.2. Initialisation

`@initialisation_phase` *`label`* Define the processes and years of the initialisation phase with label

`years` Define the number of years to run
Type: Integer
Default: No default
Value: A non-negative integer

`time_steps` Define the `@time_step` labels (in order that they are applied) in this initialisation phase
Type: String vector
Default: No default
Value: A valid time_step label, from one of `@time_step`

`lambda` Define the absolute proportional difference for assessing convergence between annual

iterations during the initialisation
Type: Constant
Default: Zero
Value: A number between 0 and 1
Note: If λ is zero, then SPM will calculate the value of the absolute proportional difference between annual iterations, but not terminate early

lambda_years Define the years to test for convergence during the initialisation
Type: Constant vector
Default: No default
Value: A vector of valid years between 1 and @initialisation_phase[label].years
Note: If λ is defined, but @initialisation_phase[label].lambda_years is not, SPM will evaluate λ for the last year in @initialisation_phase[label].years

8.3. Time-steps

@time_step *label* Define a time-step with label

processes Define the process labels, in the order that they are applied, for the time-step
Type: String vector
Default: No default
Value: Defines the labels of the processes for that time-step

8.4. Processes

The population processes available are,

- Constant recruitment process
- Beverton-Holt stock-recruit relationship recruitment process
- Local Beverton-Holt stock-recruit relationship recruitment process
- Ageing process
- Constant relationship mortality rate process
- Annually varying relationship mortality rate process
- Mortality event (as a number) process
- Mortality event (as a biomass) process
- Holling mortality rate
- Prey-suitability predation process
- Category transition process
- Category transition rate process
- Category transition by age process

The movement processes available are,

- Migration movement

- Adjacent cell movement
- Preference movement

Each type of process requires a set of subcommands and arguments specific to that process.

`@process` *`label`* Define a process with label

`type` Define the type of process
Type: String
Default: No default
Value: A valid type of process

8.4.1. `@process[label].type=constant_recruitment`

`r0` Define the total amount of recruitment at equilibrium abundance levels
Type: Estimable
Default: No default
Value: Total amount (in numbers) of recruitment applied across all categories at equilibrium abundances

`categories` Define the categories into which recruitment occurs
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`proportions` Define the proportion of recruitment that occurs into each category
Type: Estimable vector, of length `categories`
Default: No default
Value: Proportion of the annual recruitment that is applied to each category

`age` Define the age that receives recruitment
Type: Integer
Default: The minimum age of the population
Value: The age class that receives recruitment

`layer` Name of the layer used to determine where recruitment occurs
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. If a numeric layer, then recruitment is in proportion to the layer values. Note that the layer values must be non-negative

8.4.2. `@process[label].type=bh_recruitment`

`r0` Define the total amount of recruitment at equilibrium abundance levels
Type: Estimable
Default: No default
Value: Total amount (in numbers) of recruitment applied across all categories at equilibrium abundances

`categories` Define the categories into which recruitment occurs
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`proportions` Define the proportion of recruitment that occurs into each category
Type: Estimable vector, of length `@process[label].categories`
Default: No default
Value: Proportion of the annual recruitment that is applied to each category

`age` Define the age that receives recruitment
Type: Integer
Default: The minimum age of the population
Value: The age class that receives recruitment

`steepness` Define the Beverton-Holt stock recruitment relationship steepness (h) parameter
Type: Estimable
Default: 1.0
Value: Steepness value between 0.2 and 1.0

`b0` Define the `@initialisation_phase` label for the value of the derived quantity to use as the value of the spawning stock biomass (B_0)
Type: String
Default: No default
Value: Must be a valid `@initialisation_phase` label

`ssb` Define the label of the `@derived_quantity` that defines the spawning stock biomass (SSB)
Type: String
Default: No default
Value: Must be a valid `@derived_quantity` label

`ssb_offset` Define the offset (in years) for the year of the derived quantity that is to be applied as the SSB in the stock-recruit relationship
Type: Integer
Default: No default
Value: Must be a value ≥ 0

`ycs_values` YCS values
Type: Estimable vector
Default: No default
Value: Must be vector of length `@model.initial` to `@model.current`

`standardise_ycs_years` Years for which the year class strength values are defined to have mean 1.0
Type: Integer vector or integer range
Default: No default
Value: The expanded vector must have values of years between `@model.initial` and `@model.current`

`layer` Name of the layer used to determine where recruitment occurs
Type: String
Default: No layer
Value: A valid layer as defined by `@layer`. If a numeric layer, then the total recruitment R_0 in each cell is scaled to be proportional to the value of the layer in that cell.

8.4.3. `@process[label].type=local_bh_recruitment`

`r0` Define a multiplier of `r0_layer` for calculating the amount of recruitment in each cell at equilibrium abundance levels
Type: Estimable
Default: No default
Value: Multiplier of `r0_layer` to calculate the amount in each cell (in numbers) of recruitment applied across all categories at equilibrium abundances

`categories` Define the categories into which recruitment occurs
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`proportions` Define the proportion of recruitment that occurs into each category
Type: Estimable vector, of length `categories`
Default: No default
Value: Proportion of the annual recruitment that is applied to each category

`age` Define the age that receives recruitment
Type: Integer
Default: No default
Value: The age class that receives recruitment

`steepness` Define the Beverton-Holt stock recruitment relationship steepness (h) parameter
Type: Estimable
Default: 1.0
Value: Steepness value between 0.2 and 1.0

`b0` Define the `@initialisation_phase` label for the value of the derived quantity to use as the value of the spawning stock biomass (B_0) in each cell
Type: String
Default: No default
Value: Must be a valid `@initialisation_phase` label

`ssb` Define the label of the `@derived_quantity_by_cell` that defines the spawning stock

biomass (SSB) in each cell
Type: String
Default: No default
Value: Must be a valid `@derived_quantity_by_cell` label

`ssb_offset` Define the offset (in years) for the year of the derived quantity by cell that is to be applied as the SSB in the stock-recruit relationship
Type: Integer
Default: No default
Value: Must be a value ≥ 0

`ycs_values` YCS values
Type: Estimable vector
Default: No default
Value: Must be vector of length `@model.initial` to `@model.current`

`standardise_ycs_years` Years for which the year class strength values are defined to have mean 1.0
Type: Integer vector or integer range
Default: No default
Value: The expanded vector must have values of years between `@model.initial` and `@model.current`

`layer` Define the label of the layer that defines the distribution of recruitment (as a multiplier of R_0 at equilibrium abundances in each cell
Type: String
Default: No layer
Value: A valid numeric layer as defined by `@layer`. The amount of recruitment in each cell is the product of R_0 and the layer value in each cell.

8.4.4. `@process[label].type=ageing`

`categories` Define the categories that ageing is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

8.4.5. `@process[label].type=constant_mortality_rate`

`m` Define the constant mortality rate to be applied
Type: Estimable
Default: No default
Value: A positive real number

`categories` Define the categories that mortality is applied to

Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the layer
Type: String
Default: No layer
Value: A valid layer as defined by `@layer`. If a numeric layer, then mortality applied is the mortality rate multiplied by the value of the layer. Note that the layer values must be non-negative

8.4.6. `@process[label].type=constant_exploitation_rate`

`u` Define the constant exploitation rate to be applied
Type: Estimable
Default: No default
Value: A positive real number

`u_max` Define the maximum constant exploitation rate that could be applied
Type: Constant
Default: 0.99
Value: A positive real number between 0.0 And 1.0

`categories` Define the categories that mortality is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the layer
Type: String
Default: No layer
Value: A valid layer as defined by `@layer`. If a numeric layer, then mortality applied is the exploitation rate multiplied by the value of the layer. Note that the layer values must be non-negative. Note that the result of the calculation will be set to 0 if less than zero, and set to u_{max} if greater than u_{max}

8.4.7. @process[label].type=annual_mortality_rate

`years` Define the years when the mortality rates are applied
Type: Integer vector or integer range
Default: No default
Value: Valid model years

`m` Define the mortality rate to be applied for each year
Type: Estimable vector, of length `years` once expanded
Default: No default
Value: A vector of positive real numbers

`categories` Define the categories that mortality is applied to
Type: String vector
Default: No default
Value: A vector of valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector of length `categories`
Default: No default
Value: A vector of valid selectivity labels defined by `@selectivity`

`layer` Name of the multiplicative layer to be applied to M
Type: String
Default: No layer
Value: A valid numeric layer as defined by `@layer`. Note that the layer values must be non-negative

8.4.8. @process[label].type=event_mortality

`categories` Define the categories that the event mortality is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`years` Define the years where the mortality even is applied
Type: Integer vector or integer range
Default: No default
Value: Valid model years

`layers` Define the layers that specify the event mortality (as the abundance) in each year
Type: String vector, of length `years` once expanded
Default: No default
Value: Valid layers as defined by `@layer`. Note that the layer values must be non-negative

`u_max` Define the maximum exploitation rate

Type: Estimable
Default: 0.99
Value: Must be > 0 and < 1

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`penalty` Define the event mortality penalty label
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.9. `@process[label].type=biomass_event_mortality`

`categories` Define the categories that the event mortality is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`years` Define the years where the mortality event is applied
Type: Integer vector or integer range
Default: No default
Value: Valid years for the model

`layers` Define the layers that specify the event mortality (as a biomass) in each year
Type: String vector, of length `years` once expanded
Default: No default
Value: Valid layers defined by `@layer`. Note that the layer values must be non-negative

`u_max` Define the maximum exploitation rate
Type: Constant
Default: 0.99
Value: Must be > 0 and < 1

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`penalty` Define the event mortality penalty label
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.10. @process[label].type=Holling_mortality_rate

`is_abundance` Is the mortality applied as a biomass or as abundance
Type: Switch
Default: False
Value: Either True or False

`a` Define the *a* parameter of the Holling function
Type: Constant
Default: No default
Value: A positive real number

`b` Define the *b* parameter of the Holling function
Type: Constant
Default: No default
Value: A positive real number

`x` Define the type of Holling function or Michaelis-Menton function
Type: Constant
Default: default 2
Value: A positive real number, use 2 for Holling type II or 3 for Holling Type III, or other positive real value for the generalised Michaelis-Menton function

`categories` Define the categories that the Holling mortality rate is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`predator_categories` Define the categories of the predator
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`predator_selectivities` Define the selectivities applied to each predator category
Type: String vector, of length `predator_categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`u_max` Define the maximum exploitation rate
Type: Constant
Default: 0.99
Value: Must be > 0 and < 1

`penalty` Define the event mortality penalty label
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.11. `@process[label].type=Prey-suitability_predation`

`is_abundance` Is the mortality applied as a biomass or as abundance
Type: Switch
Default: False
Value: Either True or False

`consumption_rate` Define the total predator consumption rate
Type: Numeric
Default: No default
Value: A positive real number

`consumption_rate_layer` Define the label of the layer that defines the predator consumption rate in each cell
Type: String
Default: None
Value: A valid numeric layer as defined by `@layer`. The consumption rate in each cell is the product of the total consumption rate and the layer value in each cell. Note that the layer values must be non-negative.

`prey_categories` Define the prey categories that the predation mortality is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`, grouping categories into n prey groups using the $+$ symbol

`prey_selectivities` Define the selectivities applied to each prey category
Type: String vector, of length `prey_categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`electivities` Define the electivities applied to prey groups $1 \ldots n$
Type: Constant vector, of length n, the number of groups of prey in `prey_categories`
Default: No default
Value: A vector of positive real numbers, must have values that sum to 1

`predator_categories` Define the categories of the predator
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`predator_selectivities` Define the selectivities applied to each predator category
Type: String vector, of length `predator_categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`u_max` Define the maximum exploitation rate
Type: Constant
Default: 0.99
Value: Must be > 0 and < 1

`penalty` Define the process penalty label
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.12. `@process[label].type=category_state_by_age`

`category` Define the category that is the object of the process
Type: String vector
Default: No default
Value: A valid list of categories from `@model.categories`

`layer` Name of the categorical layer used to group the spatial cells for the process
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a layer of `type=categorical`

`min_age` Define the minimum age for the process
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`max_age` Define the maximum age for the process
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`N [label]` Define the following data as the number of individuals to move in each age class
Type: Estimable vectors
Default: No default
Value: The label is valid value from the associated category transition by age layer. It is followed by a vector of values giving the numbers in each age class. This subcommand is repeated for each unique value of label

8.4.13. @process[label].type=category_transition

`from` Define the categories that are the source of the transition process
Type: String vector
Default: No default
Value: A valid list of categories from `@model.categories`

`selectivities` Define the selectivities applied to the source categories
Type: String vector, of length `from`
Default: No default
Value: A valid list of selectivity labels defined by `@selectivity`

`to` Define the categories that are the sink of the transition process
Type: String vector
Default: No default
Value: A valid list of categories from `@model.categories`

`years` Define the years where the category transition is applied
Type: Integer vector or integer range
Default: No default
Value: Valid model years

`layers` Define the layers that specify the transitions (as N for each cell) in each year
Type: String vector, of length `years` once expanded
Default: No default
Value: Valid layers defined by `@layer`. Note that the layer values must be non-negative

`u_max` Define the maximum proportion of individuals that can be moved
Type: Constant
Default: 0.99
Value: Must be > 0 and ≤ 1

`penalty` Define the penalty to encourage models parameter values away from those which result in not enough individuals to move
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.14. @process[label].type=category_transition_rate

`from` Define the category that is the source of the transition process
Type: String
Default: No default
Value: A valid category from `@model.categories`

`selectivities` Define the selectivities applied to the source categories

Type: String vector, of length `from`
Default: No default
Value: A valid list of selectivity labels defined by `@selectivity`

`to` Define the category that is the sink of the transition process
Type: String
Default: No default
Value: A valid category from `@model.categories`

`proportions` Define the proportion of individuals to move
Type: Estimable
Default: No default
Value: A value ≥ 0 and ≤ 1

`layer` Name of the layer
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. If a numeric layer, then rate applied to each cell is multiplied by the value of the layer. Note that the layer values must be non-negative

8.4.15. `@process[label].type=category_transition_by_age`

`from` Define the categories that are the source of the transition process
Type: String vector
Default: No default
Value: A valid list of categories from `@model.categories`

`to` Define the categories that are the sink of the transition process
Type: String vector
Default: No default
Value: A valid list of categories from `@model.categories`

`year` Define the year when the category transition is applied
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`layer` Name of the categorical layer used to group the spatial cells for the process
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a layer of `type=categorical`

`min_age` Define the minimum age for the process
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`max_age` Define the maximum age for the process
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`N [label]` Define the following data as the number of individuals to move in each age class
Type: Estimable vectors
Default: No default
Value: The label is valid value from the associated category transition by age layer. It is followed by a vector of values giving the numbers in each age class. This subcommand is repeated for each unique value of label

`u_max` Define the maximum proportion of individuals that can be moved
Type: Constant
Default: 0.99
Value: Must be > 0 and ≤ 1

`penalty` Define the penalty to encourage models parameter values away from those which result in not enough individuals to move
Type: String
Default: No default
Value: Valid penalty label defined by `@penalty`

8.4.16. `@process[label].type=migration`

`categories` Define the categories that the migration movement event is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`proportion` Define the constant multiplier for the proportion of individuals that migrate
Type: Estimable
Default: 1.0
Value: A real number between 0 and 1, inclusive

`source_layer` Define the label of a layer that defines the source cells of the migration movement event
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`sink_layer` Define the label of a layer that defines the sink cells of the migration movement event
Type: String
Default: No default
Value: A valid layer defined by `@layer`

8.4.17. `@process[label].type=adjacent_cell`

`categories` Define the categories that the adjacent cell movement event is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Define the label of a gradient layer that defines the the relative strength of movement to adjacent cells
Type: String
Default: Default 1 in every cell, equivalent to uniform diffusion
Value: A valid layer defined by `@layer`

`proportion` Define the constant multiplier for the proportion that moves from each cell to the neighbouring cell
Type: Estimable
Default: 1.0
Value: A real number between 0 and 1, inclusive

8.4.18. `@process[label].type=preference`

`categories` Define the categories that the preference function movement is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`proportion` Define the constant multiplier for the proportion that the preference function movement is applied to
Type: Estimable
Default: 1.0
Value: A real number between 0 and 1, inclusive

`preference_functions` Define the labels of the individual preference functions that make up

the total preference function
Type: String vector
Default: No default
Value: Valid preference function labels defined by `@preference_function`

8.4.19. `@process[label].type=preference_threaded`

`categories` Define the categories that the preference function movement is applied to
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`proportion` Define the constant multiplier for the proportion that the preference function movement is applied to
Type: Estimable
Default: 1.0
Value: A real number between 0 and 1, inclusive

`preference_functions` Define the labels of the individual preference functions that make up the total preference function
Type: String vector
Default: No default
Value: Valid preference function labels defined by `@preference_function`

8.5. Preference functions

The individual preference functions available are,

- Constant
- Normal
- Double normal
- Logistic
- Inverse logistic
- Exponential
- Threshold
- Categorical
- Monotonic categorical
- Gaussian copula
- Gumbel copula
- Frank copula
- Independence copula

Each type of preference function requires a set of subcommands and arguments specific to that function.

`@preference_function` *`label`* Define a preference function with label

`type` Define the type of preference function
Type: String
Default: No default
Value: A valid type of preference function

8.5.1. `@preference_function[label].type=constant`

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

8.5.2. `@preference_function[label].type=normal`

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`mu` Defines the μ parameter of the normal preference function
Type: Estimable
Default: No default

`sigma` Defines the σ parameter of the normal preference function
Type: Estimable
Default: No default

8.5.3. @preference_function[label].type=double_normal

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`mu` Defines the μ parameter of the double-normal preference function
Type: Estimable
Default: No default

`sigma_l` Defines the σ_L parameter of the double-normal preference function
Type: Estimable
Default: No default

`sigma_r` Defines the σ_R parameter of the double-normal preference function
Type: Estimable
Default: No default

8.5.4. @preference_function[label].type=logistic

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`, with strictly positive values only

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`a50` Defines the a_{50} parameter of the logistic preference function
Type: Estimable
Default: No default

`ato95` Defines the a_{to95} parameter of the logistic preference function
Type: Estimable
Default: No default

8.5.5. @preference_function[label].type=inverse_logistic

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`, with strictly positive values only

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: No default

`a50` Defines the a_{50} parameter of the inverse-logistic preference function
Type: Estimable
Default: 1.0

`ato95` Defines the a_{to95} parameter of the inverse-logistic preference function
Type: Estimable
Default: No default

8.5.6. @preference_function[label].type=exponential

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer` of positive values only

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: No default

`lambda` Defines the λ parameter of the exponential preference function
Type: Estimable
Default: 1.0

8.5.7. @preference_function[label].type=threshold

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`n` Defines the *N* parameter of the threshold preference function
Type: Estimable
Default: No default

`lambda` Defines the λ parameter of the threshold preference function
Type: Estimable
Default: No default

8.5.8. `@preference_function[label].type=categorical`

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`category_labels` Defines the unique labels of `layer` in order of their coefficients
Type: String vector
Default: No default
Value: A complete set of unique values of labels in `layer` in order of `category_values`

`category_values` Defines the coefficients for each unique label of `layer` in order of their labels
Type: String vector
Default: No default
Value: A complete set of positive values of labels in `layer` in order of `category_labels`

8.5.9. `@preference_function[label].type=monotonic_categorical`

`layer` Defines the layer which supplies the preference function independent variable
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`alpha` Defines the multiplicative constant α
Type: Estimable
Default: 1.0

`category_labels` Defines the unique labels of `layer` in order of their coefficients
Type: String vector
Default: No default
Value: A complete set of unique values of labels in `layer` in order of `category_values`

`category_values` Defines the coefficients for each unique label of `layer` in order of their labels
Type: Numeric vector
Default: No default
Value: A complete set of positive values of labels in `layer` in order of `category_labels`

8.5.10. `@preference_function[label].type=gaussian_copula`

`rho` Defines the dependence parameter (ρ) for the copula
Type: Constant
Default: No default
Value: Any real number $>= -1$ and $<= 1$

`layers` Defines the two layers which supplies the preference function independent variables
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`pdf` Defines the two PDFs for the copula
Type: String
Default: No default
Value: Valid PDFs defined by `@pdf`

8.5.11. `@preference_function[label].type=gumbel_copula`

`rho` Defines the dependence parameter (ρ) for the copula
Type: Constant
Default: No default
Value: Any real number $>= 1$

`layers` Defines the two layers which supplies the preference function independent variables
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`pdf` Defines the two PDFs for the copula
Type: String
Default: No default
Value: Valid PDFs defined by `@pdf`

8.5.12. @preference_function[label].type=frank_copula

`rho` Defines the dependence parameter (ρ) for the copula
Type: Constant
Default: No default
Value: Any real number $! = 0$

`layers` Defines the two layers which supplies the preference function independent variables
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`pdf` Defines the two PDF for the copula
Type: String
Default: No default
Value: Valid PDFs defined by `@pdf`

8.5.13. @preference_function[label].type=independence_copula

`layers` Defines the two layers which supplies the preference function independent variables
Type: String
Default: No default
Value: A valid layer defined by `@layer`

`pdf` Defines the two PDFs for the copula
Type: String
Default: No default
Value: Valid PDFs defined by `@pdf`

8.6. Probability Density Functions

The available Probability Density Functions (PDF) types are,

- Normal
- Lognormal
- Exponential
- Uniform

`@pdf` *`label`* Define a PDF for use as a copula preference function with label

`type` Define the type of PDF
Type: String
Default: No default
Value: A valid type of PDF

8.6.1. @pdf[label].type=normal

`mu` Define the mean of the PDF
Type: Constant
Default: No default

`sigma` Define the variance of the PDF
Type: Constant
Default: No default
Value: A real number > 0

8.6.2. @pdf[label].type=lognormal

`mu` Define the mean of the PDF
Type: Constant
Default: No default

`sigma` Define the variance of the PDF
Type: Constant
Default: No default
Value: A real number > 0

8.6.3. @pdf[label].type=exponential

`lambda` Define the mean of the PDF
Type: Constant
Default: No default
Value: A real number > 0

8.6.4. @pdf[label].type=uniform

`a` Define the minimum of the PDF
Type: Constant
Default: No default
Value: A real number

`b` Define the maximum of the PDF
Type: Constant
Default: No default
Value: A real number $> a$

8.7. Layers

The available layer types are,

- Numeric
- Categorical
- Distance
- Haversine
- Abundance
- Biomass
- Abundance density
- Biomass density
- Numeric meta-layer
- Categorical meta-layer
- Derived quantity layer
- Derived quantity by cell layer

`@layer` *`label`* Define a layer function with label

`type` Define the type of layer
Type: String
Default: No default
Value: A valid type of layer

8.7.1. `@layer[label].type=numeric`

`data` Define the values of the layer
Type: Constant vector, with total length `@model.ncols` × `@model.nrows`
Default: No default
Value: A vector of values of length equal to the number of elements defined for the spatial structure

`rescale` If defined, then the values of the layer are rescaled to sum to this value
Type: Estimable
Default: No default
Note: If not defined, then the layer is not rescaled

8.7.2. `@layer[label].type=categorical`

`data` Define the values of the layer
Type: Constant vector, with total length `@model.ncols` × `@model.nrows`
Default: No default
Value: A vector of values of length equal to the number of elements defined for the spatial structure

8.7.3. @layer[label].type=distance

There are no other subcommands for `@layer[label].type=distance`.

8.7.4. @layer[label].type=haversine

`latitude` Define the layer that specifies the latitudes for each cell
Type: String
Default: No default
Condition: The argument must be a numeric layer, with values between -90 and 90.

`longitude` Define the layer that specifies the longitudes for each cell
Type: String
Default: No default
Condition: The argument must be a numeric layer, with values between 0 and 360.

8.7.5. @layer[label].type=dijkstra

There are no other subcommands for `@layer[label].type=dijkstra`.

8.7.6. @layer[label].type=haversine_dijkstra

`latitude` Define the layer that specifies the latitudes for each cell
Type: String
Default: No default
Condition: The argument must be a numeric layer, with values between -90 and 90.

`longitude` Define the layer that specifies the longitudes for each cell
Type: String
Default: No default
Condition: The argument must be a numeric layer, with values between 0 and 360.

8.7.7. @layer[label].type=abundance

`categories` Define the categories are used to calculate the abundance
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels from `@selectivity`

8.7.8. @layer[label].type=biomass

categories Define the categories are used to calculate the biomass
Type: String vector
Default: No default
Value: Valid categories from @model.categories

selectivities Define the selectivities applied to each category
Type: String vector, of length categories
Default: No default
Value: Valid selectivity labels from @selectivity

8.7.9. @layer[label].type=abundance_density

categories Define the categories are used to calculate the abundance
Type: String vector
Default: No default
Value: Valid categories from @model.categories

selectivities Define the selectivities applied to each category
Type: String vector, of length categories
Default: No default
Value: Valid selectivity labels from @selectivity

8.7.10. @layer[label].type=biomass_density

categories Define the categories are used to calculate the biomass
Type: String vector
Default: No default
Value: Valid categories from @model.categories

selectivities Define the selectivities applied to each category
Type: String vector, of length categories
Default: No default
Value: Valid selectivity labels from @selectivity

8.7.11. @layer[label].type=numeric_meta

default_layer Define the default layer to use in years or initialisation phases where it is not otherwise defined
Type: String
Default: No default
Condition: The argument must be a numeric layer

`years` Define the years that have a non-default layer
Type: Integer vector or integer range
Default: No default
Value: Must be valid model years

`layers` Define the layers for each of the years
Type: String vector, of length `year` once expanded
Default: No default
Condition: The arguments must be numeric layers

`initialisation_phases` Define the initialisation phases that have a non-default layer
Type: String vector
Default: No default
Condition: The arguments must be numeric layers

`initialisation_layers` Define the layers for each of the initialisation phases
Type: String vector, of length the number of `initialisation_phases`
Default: No default
Condition: The arguments must be numeric layers

8.7.12. `@layer[label].type=categorical_meta`

`default_layer` Define the default layer to use in years or initialisation phases where it is not otherwise defined
Type: String
Default: No default
Condition: The argument must be a categorical layer

`years` Define the years that have a non-default layer
Type: Integer vector or integer range
Default: No default
Value: Must be valid model years

`layers` Define the layers for each of the years
Type: String vector, of length `year` once expanded
Default: No default
Condition: The arguments must be categorical layers

`initialisation_phases` Define the initialisation phases that have a non-default layer
Type: String vector
Default: No default
Condition: The arguments must be categorical layers

`initialisation_layers` Define the layers for each of the initialisation phases

Type: String vector, of length the number of `initialisation_phases`
Default: No default
Condition: The arguments must be categorical layers

8.7.13. `@layer[label].type=derived_quantity`

`derived_quantity` Define the label of the `@derived_quantity` that is used as the source for the layer
Type: String
Default: No default
Value: Must be a valid `@derived_quantity` label

`year_offset` Define the offset (in years) for the year of the derived quantity that is to be applied
Type: Integer
Default: No default
Value: Must be a value ≥ 0

8.7.14. `@layer[label].type=derived_quantity_by_cell`

`derived_quantity_by_cell` Define the label of the `@derived_quantity_by_cell` that is used as the source for the layer
Type: String
Default: No default
Value: Must be a valid `@derived_quantity_by_cell` label

`year_offset` Define the offset (in years) for the year of the derived quantity by cell that is to be applied
Type: Integer
Default: No default
Value: Must be a value ≥ 0

8.8. Derived quantities by cell

The individual types of derived quantity by cell available are,

- Abundance
- Biomass

`@derived_quantity_by_cell` *`label`* Define a derived quantity by cell with label

`type` Define the type of derived quantity by cell
Type: String
Default: No default
Value: A valid type of derived quantity by cell, either `abundance` or `biomass`

8.8.1. @derived_quantity_by_cell[label].type=abundance

`categories` Define the categories are used to calculate the derived quantity by cell
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels from `@selectivity`

`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity by cell is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`time_step` Define the time-step at the end of which the derived quantity by cell is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`layer` Define the layer to be used in the calculations
Type: String
Default: None
Value: A valid numeric layer as defined by `@layer`. Note, the final values of the derived quantity by cell is the sum of the each cell is multiplied by the value of this layer.

8.8.2. @derived_quantity_by_cell[label].type=biomass

`categories` Define the categories are used to calculate the derived quantity by cell
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels from `@selectivity`

`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity by cell is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`time_step` Define the time-step at the end of which the derived quantity by cell is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`layer` Define the layer to be used in the calculations
Type: String
Default: None
Value: A valid numeric layer as defined by `@layer`. Note, the final values of the derived quantity by cell is the sum of the each cell is multiplied by the value of this layer.

8.9. Derived quantities

The individual types of derived quantities available are,

- Abundance
- Biomass

`@derived_quantity` *`label`* Define a derived quantity with label

`type` Define the type of derived quantity
Type: String
Default: No default
Value: A valid type of derived quantity, either `abundance` or `biomass`

8.9.1. `@derived_quantity[label].type=abundance`

`categories` Define the categories are used to calculate the derived quantity
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels from `@selectivity`

`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`time_step` Define the time-step at the end of which the derived quantity is calculated

Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`layer` Define the layer to be used in the calculations
Type: String
Default: None
Value: A valid numeric layer as defined by `@layer`. Note, the final value of the derived quantity is the sum of the each cell is multiplied by the value of this layer.

8.9.2. `@derived_quantity[label].type=biomass`

`categories` Define the categories are used to calculate the derived quantity
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities
Type: String vector, of length `categories`
Default: No default
Value: Valid selectivity labels from `@selectivity`

`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`time_step` Define the time-step at the end of which the derived quantity is calculated
Type: String
Default: No default
Value: A valid time-step label from `@time_step`

`layer` Define the layer to be used in the calculations
Type: String
Default: None
Value: A valid numeric layer as defined by `@layer`. Note, the final value of the derived quantity is the sum of the each cell is multiplied by the value of this layer.

8.10. Age-size relationship

The individual types of size-at-age relationships available are,

- none
- von Bertalanffy

- Schnute

@age_size *label* Define an age-size relationship with label

`type` Define the type of size-at-age relationship
Type: String
Default: No default
Value: A valid type of size-at-age relationship

8.10.1. @age_size[label].type=von_bertalanffy

`linf` Define the L_∞ parameter of the von Bertalanffy relationship
Type: Estimable
Default: No default
Value: A positive real number
Condition: Only define if using the von Bertalanffy relationship

`k` Define the k parameter of the von Bertalanffy relationship
Type: Estimable
Default: No default
Value: A positive real number
Condition: Only define if using the von Bertalanffy relationship

`t0` Define the t_0 parameter of the von Bertalanffy relationship
Type: Estimable
Default: No default
Value: A real number
Condition: Only define if using the von Bertalanffy relationship

`size_weight` Define the label of the associated size-weight relationship
Type: String
Default: No default
Value: A valid label from `@size_weight`

8.10.2. @age_size[label].type=schnute

`y1` Define the y_1 parameter of the Schnute relationship
Type: Estimable
Default: No default
Value: A positive real number
Condition: Only define if using the Schnute relationship

`y2` Define the y_2 parameter of the Schnute relationship

Type: Estimable
Default: No default
Value: A positive real number
Condition: Only define if using the Schnute relationship

`tau1` Define the τ_1 parameter of the Schnute relationship
Type: Estimable
Default: No default
Value: A real number
Condition: Only define if using the Schnute relationship

`tau2` Define the τ_2 parameter of the Schnute relationship
Type: String
Default: Normal
Value: Either normal or lognormal
Condition: Only define if using the Schnute relationship

`a` Define the *a* parameter of the Schnute relationship
Type: String
Default: Normal
Value: Either normal or lognormal
Condition: Only define if using the Schnute relationship

`b` Define the *b* parameter of the Schnute relationship
Type: String
Default: Normal
Value: Either normal or lognormal
Condition: Only define if using the Schnute relationship

`size_weight` Define the label of the associated size-weight relationship
Type: String
Default: No default
Value: A valid label from `@size_weight`

8.11. Size-weight

The individual types of size-weight relationship available are,

- None
- Basic

`@size_weight` *`label`* Define a size-weight relationship with label

`type` Define the type of relationship
Type: String
Default: No default
Value: A valid type of size-weight relationship

8.11.1. `@size_weight[label].type=none`

There are no other subcommands for `@size_weight[label].type=none`.

8.11.2. `@size_weight[label].type=basic`

`a` Define the *a* parameter of the basic size-weight relationship
Type: Estimable
Default: No default
Value: A positive real number

`b` Define the *b* parameter of the basic size-weight relationship
Type: Estimable
Default: No default
Value: A positive real number

8.12. Selectivities

The individual selectivity functions available are,

- Constant
- Knife edge
- All values
- All values bounded
- Increasing
- Logistic
- Inverse Logistic
- Logistic producing
- Double normal
- Double exponential
- Cubic spline

Each type of selectivity function requires a set of subcommands and arguments specific to that function.

`@selectivity` *`label`* Define a selectivity function with label

`type` Define the type of selectivity function
Type: String
Default: No default
Value: A valid type of selectivity function

8.12.1. @selectivity[label].type=constant

c Defines the C parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

8.12.2. @selectivity[label].type=knife_edge

e Defines the E parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

8.12.3. @selectivity[label].type=all_values

v Defines the V parameters (one for each age class) of the selectivity function
Type: Estimable vector
Default: No default
Value: A vector of positive real numbers, of length equal to the number of age classes

8.12.4. @selectivity[label].type=all_values_bounded

l Defines the L parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number

h Defines the H parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number, must be greater than L

v Defines the V parameters (one for each age class from L to H) of the selectivity function
Type: Estimable vector
Default: No default
Value: A vector of positive real numbers, of length equal to the number of age classes from L to H

8.12.5. @selectivity[label].type=increasing

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`l` Defines the L parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number

`h` Defines the H parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number, must be greater than L

`v` Defines the V parameters (one for each age class from L to H) of the selectivity function
Type: Estimable vector
Default: No default
Value: A vector of positive real numbers, of length equal to the number of age classes from L to H

8.12.6. @selectivity[label].type=logistic

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`a50` Defines the a_{50} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

`ato95` Defines the a_{to95} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

8.12.7. @selectivity[label].type=inverse_logistic

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`a50` Defines the a_{50} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

`ato95` Defines the a_{to95} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

8.12.8. `@selectivity[label].type=logistic_producing`

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`l` Defines the L parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number

`h` Defines the H parameter of the selectivity function
Type: Integer
Default: No default
Value: A positive real number, must be greater than L

`a50` Defines the a_{50} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

`ato95` Defines the a_{to95} parameter of the selectivity function
Type: Estimable
Default: No default
Value: A positive real number

8.12.9. `@selectivity[label].type=double_normal`

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`mu` Defines the μ parameter of the selectivity function
Type: Estimable
Default: No default

`sigma_l` Defines the σ_L parameter of the selectivity function
Type: Estimable
Default: No default

`sigma_r` Defines the σ_R parameter of the selectivity function
Type: Estimable
Default: No default

8.12.10. `@selectivity[label].type=double_exponential`

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`x1` Defines the x_1 parameter of the selectivity function
Type: Integer
Default: No default

`x2` Defines the x_2 parameter of the selectivity function
Type: Integer
Default: No default

`x0` Defines the x_0 parameter of the selectivity function
Type: Estimable
Default: No default

`y0` Defines the y_0 parameter of the selectivity function
Type: Estimable
Default: No default

`y1` Defines the y_1 parameter of the selectivity function
Type: Estimable
Default: No default

`y2` Defines the y_2 parameter of the selectivity function
Type: Estimable
Default: No default

8.12.11. `@selectivity[label].type=spline`

`alpha` Defines the α parameter of the selectivity function
Type: Estimable
Default: 1.0
Value: A positive real number

`knots` Defines the locations of the knots for the cubic spline function
Type: Constant vector
Default: No default
Value: A vector of real numbers of length equal to the number of values

`values` Defines the values at the knots for the cubic spline function
Type: Estimable vector
Default: No default
Value: A vector of real numbers of length equal to the number of knots

`method` The method for constraining the end values of the spline
Type: String
Default: Natural
Value: Either, `natural` (sets the 2nd derivatives to 0 at the boundaries), `fixed` (sets the 1st derivatives to zero at the boundaries), or `parabolic` sets the spline to be a parabola at the boundaries)

9. Estimation command and subcommand syntax

9.1. Estimation methods

`@estimation`

`minimiser` The label of the minimiser to use, if doing a point estimate
Type: String
Default: No default
Value: A valid label from `@minimiser`

`mcmc` The label of the MCMC to use, if doing an MCMC
Type: String
Default: No default
Value: A valid label from `@mcmc`

`profile` The labels of the profiles to use, if doing a profile
Type: String
Default: No default
Value: A valid label from `@mpd`

9.2. Point estimation

Two methods of minimising when doing a point estimate are,

- Numerical differences minimiser
- Differential evolution minimiser

Note that point estimates are required for

- MPDs
- To generate the starting values and covariance matrix for an MCMC
- To calculate the point estimates for profiles

Each type of minimiser requires a set of subcommands and arguments specific to that minimiser. Different minimisers can be called for different model runs or for different run modes (i.e., MCMC, MPDs, or profiles).

`@minimiser` *`label`* Define the an minimiser estimator with label

`type` Define the type of minimiser
Type: String
Default: numerical_differences
Value: A valid type of minimiser, either numerical_differences or de_solver

9.2.1. @minimiser[label].type=numerical_differences

`iterations` Define the maximum number of iterations for the minimiser
Type: Integer
Default: 1000
Value: A positive integer

`evaluations` Define the maximum number of evaluations for the minimiser
Type: Integer
Default: 4000
Value: A positive integer

`stepsize` Define the stepsize for the minimiser
Type: Constant
Default: 1e-6
Value: A positive real number

`tolerance` Define the convergence criteria (tolerance) for the minimiser
Type: Constant
Default: 0.002
Value: A positive real number

`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating
Type: Logical
Default: True
Value: Either true of false

9.2.2. @minimiser[label].type=de_solver

`population_size` Define the minimisers number of populations to generate
Type: Integer
Default: 25
Value: A positive integer

`crossover_probability` Define the minimisers crossover probability
Type: Integer
Default: 0.9
Value: A positive integer

`difference_scale` Define the scale of the difference of the parent candidates for the minimiser
Type: Constant
Default: 0.02
Value: A positive real number

`max_generations` Define the maximum generations for the minimiser convergence

Type: Constant
Default: 0.002
Value: A positive real number

`tolerance` Define the convergence criteria (tolerance) for the minimiser
Type: Constant
Default: 0.01
Value: A positive real number

`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating
Type: Logical
Default: True
Value: Either true of false

9.3. Markov Chain Monte Carlo (MCMC)

Only one method of carrying out MCMCs is available, Markov Chain Monte Carlo using Metropolis-Hastings

`@mcmc` *`label`* Define the MCMC estimation arguments

`type` Define the method of MCMC
Type: String
Default: metropolis_hastings
Value: A valid type of MCMC. Currently only Metropolis-Hastings is available

9.3.1. `@mcmc.type=metropolis_hastings`

`start` Covariance multiplier for the starting point of the Markov chain
Type: Constant
Default: 0
Value: If 0, defines the starting point of the chain as the point estimate. If > 0, defines the starting point as randomly generated, with covariance matrix equal to the approximate covariance (inverse Hessian) times the value of this start parameter

`length` Length of the Markov chain
Type: Integer
Default: No default
Value: Defines the length of the Markov chain (as a number of iterations)

`keep` Spacing between recorded values in the chain
Type: Integer
Default: 1
Value: Defines the spacing between recorded values in the chain. Samples from the posterior are written to file only if their sample number is evenly divisible by `keep`

`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
Type: Constant
Default: 0.8
Value: Defines the maximum correlation in the covariance matrix of the proposal distribution. Correlations greater than max_correlation are decreased to max_correlation, and those less than -max_correlation are increased to -max_correlation

`covariance_adjustment_method` Method for adjusting small variances in the covariance proposal matrix
Type: String
Default: covariance
Value: Defines the method (either correlation or covariance) for the adjusting small variances in the covariance matrix of the proposal distribution

`correlation_adjustment_diff` Minimum non-zero variance times the range of the bounds in the covariance matrix of the proposal distribution
Type: Constant
Default: 0.0001
Value: Defines the minimum non-zero variance times the difference in the bounds of each parameter in the covariance matrix of the proposal distribution

`stepsize` Initial stepsize (as a multiplier of the approximate covariance matrix)
Type: Constant
Default: $2.4d^{-0.5}$ where d is the number of estimated parameters
Value: The covariance of the proposal distribution is the approximate covariance (inverse Hessian) times this stepsize parameter

`proposal_distribution` The shape of the proposal distribution (either t or normal)
Type: String
Default: t
Value: Either `t` or `normal`. Defines whether the proposal distribution should be multivariate t rather than multivariate normal

`df` Degrees of freedom of the multivariate t proposal distribution
Type: Integer
Default: 4
Value: Defines the degrees of freedom of the multivariate t proposal distribution

`adapt_stepsize_at` Iterations in the chain to check and resize the MCMC stepsize
Type: Vector of integers
Default: no default
Value: Defines the points during the MCMC iterations to re-evaluate the MCMC stepsize argument

9.4. Profiles

`@profile` *`label`* Define the profile parameters and arguments

`parameter` Name of the parameter to be profiled
Type: String
Default: No default
Value: Defines the name of the parameter to be profiled

`steps` Number of steps (values) at which to profile the parameter
Type: Integer
Default: 10
Value: Defines the steps (number of values) at which to profile the parameter

`lower_bound` lower bound on parameter
Type: Integer
Default: No default
Value: Defines the lower bound on the range of the parameter to profile

`upper_bound` Upper bound on parameter
Type: Integer
Default: No default
Value: Defines the upper bound on the range of the parameter to profile

9.5. Defining the parameters to be estimated and their priors

`@estimate` *`parameter_name`* Estimate an estimable parameter *`parameter_name`*
The `SPM` name of the parameter to estimate
Type: string
Default: No default
Value: A valid `SPM` parameter name

`same` Names of the other parameters which are constrained to have the same value
Type: String Vector
Default: No default
Value: Defines the names of all the other parameters which are constrained to have the same value as this parameter

`lower_bound` Lower bounds on this parameter
Type: Constant vector, of length equal to the parameter length
Default: No default
Value: Defines the lower bound(s) on this parameter

`upper_bound` Upper bound on this parameter
Type: Constant vector, of length equal to the parameter length
Default: No default
Value: Defines the upper bound(s) on this parameter

`mcmc_fixed` Should this parameter be fixed during MCMC?
Type: Switch
Default: False
Value: Define this parameter as fixed during MCMC (i.e., considered a constant for the MCMC)

`type` Defines the type of prior for this parameter
Type: String
Default: No default
Value: Defines the type of prior for this parameter

9.5.1. `@estimate[label].type=uniform`

The uniform prior has no other subcommands.

9.5.2. `@estimate[label].type=uniform_log`

The uniform-log prior has no other subcommands.

9.5.3. `@estimate[label].type=normal`

`mu` Defines the mean μ of the normal prior
Type: Estimable
Default: No default
Value: Defines the mean of the normal prior

`cv` Defines the c.v. c of the normal prior
Type: Estimable
Default: No default
Value: Defines the c.v. of the normal prior

9.5.4. `@estimate[label].type=normal_by_stdev`

`mu` Defines the mean μ of the normal by standard deviation prior
Type: Estimable
Default: No default
Value: Defines the mean of the normal by standard deviation prior

`sigma` Defines the standard deviation σ of the normal by standard deviation prior
Type: Estimable
Default: No default
Value: Defines the standard deviation of the normal by standard deviation prior

9.5.5. `@estimate[label].type=lognormal`

`mu` Defines the mean μ of the lognormal prior
Type: Estimable
Default: No default
Value: Defines the mean of the lognormal prior

`cv` Defines the c.v. c of the lognormal prior
Type: Estimable
Default: No default
Value: Defines the c.v. of the lognormal prior

9.5.6. `@estimate[label].type=beta`

`a` The lower value of the range parameter A of the Beta prior
Type: Constant
Default: No default
Value: Defines the lower value of the range parameter A of the Beta prior

`b` The upper value of the range parameter B of the Beta prior
Type: Constant
Default: No default
Value: Defines the upper value of the range parameter B of the Beta prior

`mu` Defines the mean μ of the Beta prior
Type: Estimable
Default: No default
Value: Defines the mean of the Beta prior

`sigma` Defines the standard deviation σ of the Beta prior
Type: Estimable
Default: No default
Value: Defines the standard deviation of the Beta prior

9.6. Defining catchability constants

`@catchability` *`label`* Define a catchability constant with `label`

`q` Value of the q parameter
Type: Estimable
Default: No default
Value: Defines the value of the catchability q parameter, a real positive number

9.7. Defining penalties

`@penalty` *`label`* Define a penalty with `label`

`log_scale` Defines if the penalty in calculated in log space
Type: Switch
Default: False
Value: Defines if the value of the penalty is calculated as the squared difference or the squared difference in log space

`multiplier` Penalty multiplier
Type: Constant
Default: 1.0
Value: Defines the penalty multiplier

10. Observation command and subcommand syntax

10.1. Observation types

The observation types available are,

Observations of proportions of individuals by age class

Observations of proportions of individuals between categories within each age class

Relative and absolute abundance observations

Relative and absolute biomass observations

Relative proportions present observations

Each type of observation requires a set of subcommands and arguments specific to that process.

`@observation` *`label`* Define an observation

`type` Define the type of observation
Type: String
Default: No default
Value: A valid type of observation

10.1.1. `@observation[label].type=proportions_at_age`

`year` Define the year that the observation applies to
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step that the observation applies to
Type: Integer
Default: No default
Value: A valid time-step

`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
Type: String
Default: mean
Value: Either `mean` for the weighted mean, or `difference` for the difference between the start and end state

`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
Type: Constant
Default: 1.0
Value: A real number between 0 and 1, inclusive

`categories` Define the categories

Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`min_age` Define the minimum age for the observation
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`max_age` Define the maximum age for the observation
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`age_plus_group` Define is the the maximum age for the observation is a plus group
Type: Switch
Default: True
Value: Either true or false

`ageing_error` Define the label of the ageing-error matrix to be applied (if any)
Type: String
Default: No default
Value: A valid ageing error label

`layer` Name of the categorical layer used to group the spatial cells for the observation
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a layer of `type=categorical`

`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
Type: Constant vector
Default: No default
Value: The label is valid value from the associated observation layer. It is followed by a vector of values giving the proportions at age. This subcommand is repeated for each unique value of label

`tolerance` Define the tolerance on the sum-to-one error check in `SPM`
Type: Constant
Default: 0.001
Value: The tolerance on the sum to one error check. If $abs(1-\sum O_i) >$ `tolerance` then `SPM` will report an error

`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
Type: Constant
Default: No default
Value: A valid value from the associated observation layer. This subcommand is repeated for each unique value of label

`likelihood` Define the likelihood for the observation
Type: String
Default: No default
Value: A valid likelihood; either Dirichlet, multinomial, or lognormal

`delta` Define the delta robustifying constant for the likelihood
Type: Constant
Default: 1e-11
Value: A non-negative real number

`process_error` Define the process error term
Type: Estimable
Default: No process error
Value: A non-negative real number. A zero (0) can be used to specify no process error

`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation
Type: String
Default: No default
Value: A valid likelihood, except not `none`. Note that this command is ignored if the observation is not a pseudo-observation

10.1.2. `@observation[label].type=proportions_by_category`

`year` Define the year that the observation applies to
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step that the observation applies to
Type: Integer
Default: No default
Value: A valid time-step

`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
Type: String
Default: mean
Value: Either `mean` for the weighted mean, or `difference` for the difference between the start and end

state

`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
Type: Constant
Default: 1.0
Value: A real number between 0 and 1, inclusive

`categories` Define the categories that make up the numerator of the observation
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`categories2` Define the categories that, in combination with the numberator categories, make up the denominator
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`selectivities2` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`min_age` Define the minimum age for the observation
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`max_age` Define the maximum age for the observation
Type: Integer
Default: No default
Value: A valid age in the range `@model.min_age` and `@model.max_age`

`age_plus_group` Define is the the maximum age for the observation is a plus group
Type: Switch
Default: True
Value: Either true or false

`ageing_error` Define the label of the ageing-error matrix to be applied

Type: String
Default: No default
Value: A valid ageing error label

`layer` Name of the categorical layer used to group the spacial cells for the observation
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a categorical layer

`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
Type: Constant vector
Default: No default
Value: The label is valid value from the associated observation layer. It is followed by a vector of values giving the proportions at age. This subcommand is repeated for each unique value of label

`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
Type: Constant
Default: No default
Value: A valid value from the associated observation layer. This subcommand is repeated for each unique value of label

`likelihood` Define the likelihood for the observation
Type: String
Default: No default
Value: A valid likelihood; either binomial or binomial-approx

`delta` Define the delta robustifying constant for the likelihood
Type: Constant
Default: 1e-11
Value: A non-negative real number

`process_error` Define the process error term
Type: Estimable
Default: No process error
Value: A non-negative real number. A zero (0) can be used to specify no process error

`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation
Type: String
Default: No default
Value: A valid likelihood, except not `none`. Note that this command is ignored if the observation is not a pseudo-observation

10.1.3. @observation[label].type=abundance

`year` Define the year that the observation applies to
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step that the observation applies to
Type: Integer
Default: No default
Value: A valid time-step

`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
Type: String
Default: mean
Value: Either `mean` for the weighted mean, or `difference` for the difference between the start and end state

`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
Type: Constant
Default: 1.0
Value: A real number between 0 and 1, inclusive

`catchability` Define the catchability constant label for the observation
Type: String
Default: No default
Value: A valid `@catchability` label

`categories` Define the categories for which the observations occur
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the categorical layer used to group the spacial cells for the observation
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a categorical layer

`obs [label]` Define the following data as observations for the categorical layer with value

`[label]`
Type: Constant
Default: No default
Value: The label is valid value from the associated observation layer. It is followed by a value giving the abundance. This subcommand is repeated for each unique value of label

`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
Type: Constant
Default: No default
Value: A valid value from the associated observation layer. This subcommand is repeated for each unique value of label

`likelihood` Define the likelihood for the observation
Type: String
Default: No default
Value: A valid likelihood; either lognormal or normal

`delta` Define the delta robustifying constant for the likelihood
Type: Constant
Default: 1e-11
Value: A non-negative real number

`process_error` Define the process error term
Type: Estimable
Default: No process error
Value: A non-negative real number. A zero (0) can be used to specify no process error

`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation
Type: String
Default: No default
Value: A valid likelihood, except not `none`. Note that this command is ignored if the observation is not a pseudo-observation

10.1.4. `@observation[label].type=biomass`

`year` Define the year that the observation applies to
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step that the observation applies to
Type: Integer
Default: No default
Value: A valid time-step

`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
Type: String
Default: mean
Value: Either `mean` for the weighted mean, or `difference` for the difference between the start and end state

`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
Type: Constant
Default: 1.0
Value: A real number between 0 and 1, inclusive

`catchability` Define the catchability constant label for the observation
Type: String
Default: No default
Value: A valid `@catchability` label

`categories` Define the categories for which the observations occur
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the categorical layer used to group the spacial cells for the observation
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a categorical layer

`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
Type: Constant vector
Default: No default
Value: The label is valid value from the associated observation layer. It is followed by a value giving the biomass. This subcommand is repeated for each unique value of label

`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
Type: Constant
Default: No default
Value: A valid value from the associated observation layer. This subcommand is repeated for each unique value of label

`likelihood` Define the likelihood for the observation
Type: String
Default: No default
Value: A valid likelihood; either lognormal or normal

`delta` Define the delta robustifying constant for the likelihood
Type: Constant
Default: 1e-11
Value: A non-negative real number

`process_error` Define the process error term
Type: Estimable
Default: No process error
Value: A non-negative real number. A zero (0) can be used to specify no process error

`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation
Type: String
Default: No default
Value: A valid likelihood, except not `none`. Note that this command is ignored if the observation is not a pseudo-observation

10.1.5. `@observation[label].type=presence`

`year` Define the year that the observation applies to
Type: Integer
Default: No default
Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step that the observation applies to
Type: Integer
Default: No default
Value: A valid time-step

`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
Type: String
Default: mean
Value: Either `mean` for the weighted mean, or `difference` for the difference between the start and end state

`proportion_time_step` Define the interpolated proportion through the time-step for calculating

the expected value of the observation
Type: Constant
Default: 1.0
Value: A real number between 0 and 1, inclusive

`catchability` Define the catchability constant label for the observation
Type: String
Default: No default
Value: A valid `@catchability` label

`categories` Define the categories for which the observations occur
Type: String vector
Default: No default
Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category
Type: String vector
Default: No default
Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the categorical layer used to group the spacial cells for the observation
Type: String
Default: No default
Value: A valid layer as defined by `@layer`. Must be a categorical layer

`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
Type: Constant vector
Default: No default
Value: The label is valid value from the associated observation layer. It is followed by a value giving the proportion. This subcommand is repeated for each unique value of label

`error_value [label]` Define the following data as error values (e.g., N for binomial likelihoods) for the categorical layer with value `[label]`
Type: Constant
Default: No default
Value: A valid value from the associated observation layer. This subcommand is repeated for each unique value of label

`likelihood` Define the likelihood for the observation
Type: String
Default: No default
Value: A valid likelihood; binomial is the only valid likelihood

`delta` Define the delta robustifying constant for the likelihood

Type: Constant
Default: 1e-11
Value: A non-negative real number

`process_error` Define the process error term
Type: Estimable
Default: No process error
Value: A non-negative real number. A zero (0) can be used to specify no process error

`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation
Type: String
Default: No default
Value: A valid likelihood, except not `none`. Note that this command is ignored if the observation is not a pseudo-observation

10.2. Defining ageing error

Three methods for including ageing error into estimation with observations are,

- None
- Normal
- Off-by-one

Each type of ageing error requires a set of subcommands and arguments specific to its type.

`@ageing_error` *`label`* Define ageing error with `label`

`type` The type of ageing error
Type: String
Default: No default
Value: Defines the type of ageing error to use

10.2.1. `@ageing_error[label].type=none`

The `@ageing_error[label].type=none` has no other subcommands.

10.2.2. `@ageing_error[label].type=normal`

`cv` Parameter of the normal ageing error model
Type: Constant
Default: No default
Value: Define the c.v. of misclassification

`k` The *k* parameter of the normal ageing error model

Type: Integer
Default: 0
Value: *cv* defines the proportions of misclassification down and up using the normal model. *k* defines the minimum age of individuals which can be misclassified, e.g., individuals under age *k* have no ageing error

10.2.3. `@ageing_error[label].type=off_by_one`

`p1` The p_1 parameter of the off-by-one ageing error model
Type: Constant
Default: No default
Value: p_1 and p_2 define the proportions of misclassification down and up by 1 year respectively. *k* defines the minimum age of individuals which can be misclassified, e.g. individuals under age *k* have no ageing error

`p2` The p_2 parameter of the off-by-one ageing error model
Type: Constant
Default: No default
Value: p_1 and p_2 define the proportions of misclassification down and up by 1 year respectively. *k* defines the minimum age of individuals which can be misclassified, e.g., individuals under age *k* have no ageing error

`k` The *k* parameter of the off-by-one ageing error model
Type: Integer
Default: 0
Value: p_1 and p_2 define the proportions of misclassification down and up by 1 year respectively. *k* defines the minimum age of individuals which can be misclassified, e.g., individuals under age *k* have no ageing error

11. Report command and subcommand syntax

11.1. Available reports

The report types available are,

1. Print the map (i.e., row and column labels of each spatial cell) of the spatial structure
2. Print the partition at a specific time-step for any number of years
3. Print the biomass of the partition at a specific time-step for any number of years
4. Print the partition at the end of an initialisation
5. Print a summary of a process
6. Print a summary of a preference function
7. Print a derived quantity
8. Print a derived quantity by cell
9. Print a summary of the estimated parameters
10. Print the estimated parameters in a vector format (suitable for use with `spm -i`)
11. Print the objective function values
12. Print the covariance matrix
13. Print an observation values, fits, and residuals
14. Print a simulated observation suitable for use in a `SPM` input configuration file.
15. Print the ageing error misclassification matrix
16. Print a layer
17. Print a derived view via a categorical layer
18. Print a selectivity's values
19. Print the random number seed
20. Print the age-size relationship
21. Print the age-weight relationship
22. Print the size-weight relationship
23. Print the results of an MCMC
24. Print the MCMC samples as they are calculated
25. Print the MCMC objective function values as they are calculated

Each type of report requires a set of subcommands and arguments specific to that report.

11.2. Report commands and subcommands

`@report` *`label`* Define an output report

`type` Define the type of report
Type: String
Default: No default
Value: A valid type of report

11.2.1. @report[label].type=spatial_map

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out
Note: Some reports *require* that a file name is provided.

11.2.2. @report[label].type=partition

years Define the years that the partition report applies to
Type: Integer vector or integer range
Default: All model years
Value: Valid model years between `@model.initial_year` and `@model.current_year`

time_step Define the time-step that the partition report applies to
Type: Integer
Default: No default
Value: A valid time-step

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.3. @report[label].type=partition_biomass

years Define the years that the partition_biomass report applies to
Type: Integer vector or integer range
Default: All model years
Value: Valid model years between `@model.initial_year` and `@model.current_year`

time_step Define the time-step that the partition_biomass report applies to
Type: Integer
Default: No default
Value: A valid time-step

file_name Define the name of the output file where the report is written

Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.4. `@report[label].type=initialisation`

`initialisation_phase` Define the phase of initialisation that the partition report applies to
Type: string
Default: No default
Value: A valid phase label, from `@initialisation_phase`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.5. `@report[label].type=process`

`process` Define the label of the process to summarise
Type: String
Default: No default
Value: A valid label from `@process`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.6. @report[label].type=preference_function

preference_function Define the label of the preference function to summarise
Type: String
Default: No default
Value: A valid label from @preference_function

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.7. @report[label].type=derived_quantity

derived_quantity Define the label of the derived quantity to print
Type: String
Default: No default
Value: A valid label from @derived_quantity

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.8. @report[label].type=derived_quantity_by_cell

derived_quantity_by_cell Define the label of the derived quantity by cell to print
Type: String
Default: No default
Value: A valid label from @derived_quantity_by_cell

`initialisation` Specify if the derived quantity by cell values for each year of the initialisation phases should be also be output
Type: Switch
Default: False
Value: Either True or False

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.9. `@report[label].type=estimate_summary`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.10. `@report[label].type=estimate_value`

Prints the estimated parameters in a format suitable for use with `spm -i`.

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`header` Specify if the output contains the standard `SPM` style header at the start of the output
Type: Switch
Default: True if `file_name` is not defined, otherwise False.
Value: Either True or False. Use False if you want to output this to a file in a format that it can be read by SPM as a parameter input file

`overwrite` Specify if any previous file with the same name as the output file should be

overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.11. @report[label].type=objective_function

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.12. @report[label].type=covariance

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.13. @report[label].type=observation

`observation` Define the label of the observation to print
Type: String
Default: No default
Value: A valid label from `@Observation`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be

overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.14. @report[label].type=simulated_observation

`observation` Define the label of the observation from which to simulate values
Type: String
Default: No default
Value: A valid label from `@observation`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.15. @report[label].type=ageing_error

`ageing_error` Define the label of the ageing_error misclassification matrix
Type: String
Default: No default
Value: A valid label from `@ageing_error`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.16. @report[label].type=layer

`layer` Define the label of the layer to print
Type: String
Default: No default
Value: A valid label from `@layer`

`years` Define the years for the printing of the layer
Type: Integer vector or integer range
Default: The first year of the model, `@model.initial_year`
Value: Valid model years between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step for the printing of the layer
Type: Integer
Default: The first time-step of the model
Value: A valid time-step

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.17. @report[label].type=layer_derived_view

`layer` Define the label of the layer to print
Type: String
Default: No default
Value: A valid label from `@layer` of type `categorical`

`years` Define the years for the printing of the layer
Type: Integer vector or integer range
Default: The first year of the model, `@model.initial_year`
Value: Valid model years between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step for the printing of the layer
Type: Integer
Default: The first time-step of the model
Value: A valid time-step

`file_name` Define the name of the output file where the report is written

Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.18. `@report[label].type=selectivity`

`selectivity` Define the label of the selectivity to print
Type: String
Default: No default
Value: A valid label from `@selectivity`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.19. `@report[label].type=random_number_seed`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.20. @report[label].type=age_size

age_size Define the label of the age-size relationship
Type: String
Default: No default
Value: A valid label from @age_age

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.21. @report[label].type=size_weight

age_size Define the label of the age-size command that specifies the size-weight relationship
Type: String
Default: No default
Value: A valid label from @age_size

sizes Define the sizes for which to report the mean weights
Type: Constant vector
Default: No default
Value: Values of sizes to report the mean weight

file_name Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.22. @report[label].type=age_weight

`age_size` Define the label of the age-size relationship
Type: String
Default: No default
Value: A valid label from `@age_size`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.23. @report[label].type=MCMC

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. If not supplied, then output is directed to the standard out

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.24. @report[label].type=MCMC_samples

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. Must be supplied.

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

11.2.25. `@report[label].type=MCMC_objectives`

`file_name` Define the name of the output file where the report is written
Type: String
Default: No default
Value: A valid file name. Must be supplied.

`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to
Type: Switch
Default: True
Value: Either True or False

12. Other commands and subcommands

`@include` *`file`* Include an external file

`file` The name of the external file to include

Type: string
Default: No default
Value: A valid external file
Condition: The file name must be enclosed in double quotes
Example: `@include "my_file.txt"`
Note: `@include` does not denote the end of the previous command block as is the case for all other commands

13. Examples

We provide some simple examples to illustrate the syntax of the input configuration file, provide an introduction to the command calls to SPM, and show respective outputs. Note, we reproduce only a subset of the input configuration files — for more detail, see the example files.

Two examples are provided. The first implements a simple population that resides in a single cell, without movement. The second example has a similar population structure inside a 10×6 spatial matrix, and moves immature and mature fish according to preference function movement process.

13.1. An example of a simple 1×1 non-spatial model

This example implements a very simple single spatial cell model (i.e., a model with no movement, and the entire population enclosed within a 1×1 spatial structure) with recruitment, maturation, natural and exploitation mortality, and an annual age increment. The population structure has ages $1-50^{+}$ with categories labelled immature and mature.

The model was initialised over a 200 year period, and applies the following processes,

1. A Beverton-Holt recruitment process, recruiting a constant number of individuals to the first age class (i.e., $age = 1$) in the category labelled immature.
2. A maturation process, where individuals are moved from the immature to the mature categories with a logistic-producing selectivity labelled 'maturation'.
3. A constant mortality process representing natural mortality, applied as two repeats of the 'halfM' process. (Half M used so-as to be able to mimic a $\frac{1}{2}M + F + \frac{1}{2}M$ natural and fishing exploitation set of processes after initialisation.)
4. An ageing process, where all individuals are aged by one year, and with a plus group accumulator age class at $age = 50$.

A second phase of initialisation, of period one year, is applied to allow external validation that the initialisation process has stabilised the population to equilibrium (i.e., by confirming that there is no or at least only a small difference in the partition at the end of first and second phases).

Following initialisation, the model runs from the years 1994 to 2007 iterating through two time-steps. The first time-step applies processes of recruitment, maturation, and $\frac{1}{2}M + F + \frac{1}{2}M$ processes. The exploitation process (fishing) is applied in the years 1998–2007, with catches defined by the layers Fishing_1998 to Fishing_2007.

The second time-step applies an age increment.

The first 50 lines of the main section of the input configuration file are,

```
#########################################################################
# Example 1: Simple 1x1 spatial model without movement
#########################################################################

# Model structure
@model
nrows 1
ncols 1
layer Base
categories immature mature
min_age 1
max_age 35
age_plus_group True
```

```
initial_year 1994
current_year 2007
cell_length 1
initialisation_phases Phase1
time_steps step_one step_two
age_size VB VB

@age_size VB
type von_bertalanffy
size_weight basic
k 0.093
t0 -0.256
linf 169.07

@size_weight basic
type basic
a 1.387e-008
b 2.965

# Initalisation
@initialisation_phase Phase1
years 200
time_steps Phase1a Phase1b
lambda 1e-10
lambda_years 100 125 150 175 200

@time_step Phase1a
processes BHrecruitment Maturation halfM

@time_step Phase1b
processes halfM Ageing

@time_step Phase1c
processes halfM Ageing

# Annual cycle
@time_step step_one
```

The input configuration file includes definitions of required layers and the estimation, observation, and report parameters as external files.

To carry out a run of the model (to verify that the model runs without any syntax errors), use the command `spm -r -c config.spm`. Note that as `SPM` looks for a file named `config.spm` by default, we can simplify the command to `spm -r`.

To run an estimation, and hence estimate the parameters defined in the file `estimation.spm` (the catchability constant q, recruitment R_0, and the selectivity parameters a_{50} and a_{to95}), use `spm -e`. Here, we have piped the output to `estimate.log` using the command `spm -e > estimate.log`. `SPM` reports a the results of each iteration of the estimation, and ends with successful convergence,

```
Convergence was successful
Total elapsed time: 1 second
```

The main part of the output from the estimation run is summarised in the file `estimate.log`, and the final objective function is,

```
[Objective_Score]
report.type: objective_function
obs->CAA-year-1998: 26.9697
obs->CAA-year-1999: 33.097
obs->CAA-year-2000: 27.9816
```

```
obs->CAA-year-2001: 27.2661
obs->CAA-year-2002: 26.1045
obs->CAA-year-2003: 32.2496
obs->CAA-year-2004: 31.8422
obs->CAA-year-2005: 28.0624
obs->CAA-year-2006: 26.7884
obs->CAA-year-2007: 28.982
obs->CPUE-1998: -1.19322
obs->CPUE-1999: -1.26801
obs->CPUE-2000: -0.801419
obs->CPUE-2001: 1.73287
obs->CPUE-2002: -1.24005
obs->CPUE-2003: -1.61189
obs->CPUE-2004: -1.42109
obs->CPUE-2005: -1.4817
obs->CPUE-2006: -1.61507
obs->CPUE-2007: -1.1968
prior->catchability[CPUEq].q: 0
prior->process[BHrecruitment].r0: 0
prior->selectivity[FishingSel].a50: 0
prior->selectivity[FishingSel].ato95: 0
total_score: 279.247
*end
```

with parameter estimates,

```
catchability[CPUEq].q process[BHrecruitment].r0 selectivity[FishingSel].a50 selectivity[
    FishingSel].ato95
0.000132488 344098 8.62018 3.29177
```

A profile on the R_0 parameter can also be run, using `spm -p > profile.log`. See the examples folder for an example of the output.

Simulations were run using command `spm -s 10 > simulations.log`. The first 20 lines of the first simulation are,

```
#[CAA-1998]
#report.type: simulated_observation
#observation.label: CAA-year-1998
@observation CAA-year-1998
age_plus_group True
ageing_error normal
categories immature + mature
delta 1e-11
layer SSRU
likelihood multinomial
max_age 35
min_age 1
process_error 25
selectivities FishingSel FishingSel
time_step step_one
type proportions_at_age
year 1998
obs All 0 0 0 0 0 0.038461538461538464 0.076923076923076927 0.038461538461538464
    0.076923076923076927 0 0.038461538461538464 0.038461538461538464 0.038461538461538464
    0.15384615384615385 0.076923076923076927 0 0 0.076923076923076927 0.038461538461538464
    0.038461538461538464 0 0.076923076923076927 0 0.038461538461538464 0.038461538461538464
    0.038461538461538464 0 0 0 0 0 0 0 0 0.076923076923076927
error_value All 343
#end
```

An estimate of the posterior distribution can be found using the command `spm -m -g 0 >`

MCMC.log. The first set of output describes the covariance matrix and the MCMC proposal matrix.

```
SPM (Spatial Population Model)
Call: spm -m -q -g 0
Date: Thu Nov 19 22:42:33 2015
v1.1-2015-11-19 (rev. 1272). Copyright (c) 2008-2015, NIWA
User name: dunn
Machine name: NIWA-34286 (Windows_NT, PID=6480)

[MCMC]
report.type: MCMC
Original covariance matrix:
1.4961e-009 -2.82829 6.40938e-006 3.33788e-006
-2.82829 5.83449e+009 -6405.04 -3098.07
6.40938e-006 -6405.04 0.211802 0.158556
3.33788e-006 -3098.07 0.158556 0.24515
Proposal distribution covariance matrix:
1e-006 -61.1071 0.000165705 8.6296e-005
-73.1213 5.83449e+009 -6405.04 -3098.07
0.000165705 -6405.04 0.211802 0.158556
8.6296e-005 -3098.07 0.158556 0.24515
```

Following this, the log file contains the iterations, objective functions values, MCMC acceptance rate, and the step size.

```
MCMC objective function values:
iteration score penalty prior likelihood acceptance_rate acceptance_rate_since_adapt stepsize
0 279.247 0 0 279.247 0 0 1.2
500 282.374 0 0 523.254 0.012 0.012 1.2
1000 280.438 0 0 634.86 0.011 0.011 1.2
1500 281.066 0 0 286.627 0.0646667 0.172 0.055
2000 280.179 0 0 284.741 0.0895 0.168 0.055
2500 279.999 0 0 314.175 0.1104 0.176667 0.055
3000 280.604 0 0 295.179 0.122 0.1775 0.055
3500 279.354 0 0 448.064 0.125714 0.1716 0.055
4000 279.809 0 0 279.809 0.12975 0.169333 0.055
4500 280.233 0 0 316.346 0.136889 0.172857 0.055
5000 279.636 0 0 298.516 0.1414 0.174 0.055
5500 281.274 0 0 282.435 0.144 0.173556 0.055
6000 279.555 0 0 287.082 0.145833 0.1728 0.055
6500 281.763 0 0 325.234 0.147077 0.171818 0.055
7000 281.46 0 0 349.821 0.149571 0.172667 0.055
7500 282.851 0 0 292.305 0.1508 0.172308 0.055
8000 279.781 0 0 296.092 0.153375 0.173714 0.055
8500 279.386 0 0 293.882 0.155882 0.1752 0.055
9000 281.886 0 0 282.742 0.157111 0.175375 0.055
9500 280.238 0 0 281.397 0.160211 0.177765 0.055
10000 280.347 0 0 423.264 0.1616 0.178333 0.055
10500 281.102 0 0 440.648 0.166 0.254 0.0408681
11000 280.377 0 0 350.777 0.168545 0.238 0.0408681
11500 280.154 0 0 280.154 0.172609 0.246 0.0408681
```

The actual MCMC samples are listed at the end of the file.

```
MCMC samples:
catchability[CPUEq].q process[BHrecruitment].r0 selectivity[FishingSel].a50 selectivity[
    FishingSel].ato95
0.000132488 344098 8.62018 3.29177
0.000173174 280791 7.98245 2.73958
0.000122768 389775 9.18132 3.5994
0.000131988 380331 8.79455 3.6896
0.000114202 370440 8.30919 2.71256
0.000129867 370192 9.0649 3.85979
```

```
0.000133544 353899 8.48141 2.73513
0.000121365 372519 8.54416 3.24024
0.000119066 362792 8.32842 2.80378
0.000121409 361322 8.87031 3.88152
0.000115728 365995 8.31655 2.92748
0.000127486 384377 8.43292 3.18015
0.000113634 377247 8.31283 3.14061
0.000101929 379279 8.48682 3.65527
0.000121086 361477 8.89112 3.05047
0.000159626 283860 7.79531 2.89878
0.000129861 335655 8.61404 3.56582
0.000130395 359980 8.74156 3.38874
0.000121371 382987 9.36395 3.59125
0.000137396 322513 8.48643 2.80976
0.000128102 356723 8.32132 3.3633
0.000113157 356562 8.64887 3.63228
0.000157683 328223 8.88792 3.61253
0.000159144 308982 8.52651 3.3771
```

13.2. An example of a simple 10×6 spatial model with movement

This example implements a 10×6 spatial cell model, with preference function movement processes. The population has recruitment, maturation, natural and exploitation mortality, and annual age increment processes, with ages $1 - 50^+$ and categories labelled immature and mature.

The model was initialised over a 100 year period, and applies the following population processes;

1. A Beverton-Holt recruitment process, recruiting a constant number of individuals to the first age class (i.e., $age = 1$) in the category labelled immature.
2. A maturation process, where individuals are moved from the immature to the mature categories with a logistic-producing selectivity labelled 'maturation'.
3. A constant mortality process representing natural mortality.
4. An ageing process, where all individuals are aged by one year, and with a plus group accumulator age class at $age = 50$.

The second phase of initialisation also covers 100 years, but includes movement processes in addition to the population processes above;

1. For immature fish, a preference function process that applies movement according to distance, depth, and latitude.
2. For mature fish, a preference function process that applies movement also according to distance, depth, and latitude, but with different functional forms and parameters to the immature movement.

A third phase of initialisation, of period one year, is applied to allow external validation that the initialisation process has stabilised the population to equilibrium (i.e., by confirming that there is no or at least only a small difference in the partition at the end of second and third phases).

Following initialisation, the model runs from the years 1994 to 2007 iterating through two time-steps. The first time-step applies processes of recruitment, maturation, and $M + F$ processes. The exploitation process (fishing) is applied in the years 1998–2007, with catches defined by the layers Fishing_1998 to Fishing_2007. Movement is applied along with ageing in the second time-step.

The first 60 lines of the main section of the input configuration file are,

```
##########################################################################
# Example 2: 10x6 spatial model with preference movement processes
##########################################################################

# Model Structure
@model
nrows 6
ncols 10
layer Base
categories immature mature
min_age 1
max_age 30
age_plus_group True
initialisation_phases Phase1 Phase2
initial_year 1995
current_year 2007
cell_length 100
time_steps one two
age_size none none

@age_size none
type none
size_weight none

@size_weight none
type none

# Initialisation
@initialisation_phase Phase1
years 100
time_steps initial_step_one
lambda 1e-10
lambda_years 10 20 30 40 50 60 70 80 90 100

@initialisation_phase Phase2
years 100
time_steps initial_step_two
lambda 1e-30
lambda_years 10 20 30 40 50 60 70 80 90 100

@time_step initial_step_one
processes BHrecruitment Maturation M Ageing

@time_step initial_step_two
processes BHrecruitment Maturation M MoveImmature MoveMature Ageing

# Annual Cycle
@time_step one # Summer
processes BHrecruitment Maturation M Fishing

@time_step two # Winter
processes MoveImmature MoveMature Ageing

# Derived quantities
@derived_quantity SSB
type abundance
time_step one
initialisation_time_steps initial_step_one initial_step_two
categories mature
selectivities One
```

The input configuration file includes definitions of required layers and the estimation, observation, and report parameters as external files.

To carry out a run of the model (to verify that the model runs without any syntax errors), use the command `spm -r -c config.spm`. Note that as `SPM` looks for a file named `config.spm` by default, we can simplify the command to `spm -r`.

To run an estimation, and hence estimate the parameters defined in the file `estimation.spm` (the catchability constant q, recruitment R_0, and the selectivity parameters a_{50} and a_{to95}), use `spm -e`. Here, we have piped the output to `estimate.log` using the command `spm -e > estimate.log`. `SPM` reports a the results of each iteration of the estimation, and ends with successful convergence,

```
Convergence was successful
Total elapsed time: 30 minutes
```

The main part of the output from the estimation run is summarised in the file `estimate.log`, and the final objective function is,

```
[Objective_Score]
report.type: objective_function
obs->mature-2002: 102.812
obs->mature-2003: 93.2912
obs->mature-2004: 123.13
obs->mature-2005: 298.976
obs->mature-2006: 131.934
obs->mature-2007: 192.088
obs->CAA-1998: 243.084
obs->CAA-1999: 201.926
obs->CAA-2000: 115.418
obs->CAA-2001: 354.217
obs->CAA-2002: 349.143
obs->CAA-2003: 308.464
obs->CAA-2004: 476.683
obs->CAA-2005: 489.697
obs->CAA-2006: 316.398
obs->CAA-2007: 419.028
obs->cpue-1998: 33.9539
obs->cpue-1999: 16.2325
obs->cpue-2000: 1.72552
obs->cpue-2001: 33.3928
obs->cpue-2002: 59.2937
obs->cpue-2003: 32.1744
obs->cpue-2004: 30.0503
obs->cpue-2005: 25.2473
obs->cpue-2006: 81.4063
obs->cpue-2007: 95.7055
prior->catchability[CPUEq].q: 0
prior->preference_function[ImmatureDistance].lambda: 0
prior->preference_function[ImmatureDepth].a50: 0
prior->preference_function[ImmatureDepth].ato95: 0
prior->preference_function[ImmatureLatitude].a50: 0
prior->preference_function[ImmatureLatitude].ato95: 0
prior->preference_function[MatureDistance].lambda: 0
prior->preference_function[MatureDepth].mu: 0
prior->preference_function[MatureDepth].sigma_l: 0
prior->preference_function[MatureDepth].sigma_r: 0
prior->preference_function[MatureLatitude].a50: 0
prior->preference_function[MatureLatitude].ato95: 0
prior->selectivity[FishingSel].a50: 0
prior->selectivity[FishingSel].ato95: 0
prior->selectivity[Maturation].a50: 0
```

```
prior->selectivity[Maturation].ato95: 0
prior->process[BHrecruitment].r0: 0
total_score: 4625.47
*end
```

with parameter estimates,

```
catchability[CPUEq].q preference_function[ImmatureDistance].lambda preference_function[
    ImmatureDepth].a50 preference_function[ImmatureDepth].ato95 preference_function[
    ImmatureLatitude].a50 preference_function[ImmatureLatitude].ato95 preference_function[
    MatureDistance].lambda preference_function[MatureDepth].mu preference_function[MatureDepth
    ].sigma_l preference_function[MatureDepth].sigma_r preference_function[MatureLatitude].a50
     preference_function[MatureLatitude].ato95 selectivity[FishingSel].a50 selectivity[
    FishingSel].ato95 selectivity[Maturation].a50 selectivity[Maturation].ato95 process[
    BHrecruitment].r0
0.000110781 0.0129189 4535.66 4290.64 70.4152 7.42919 0.00650572 4997.67 2399.27 9893.19
    61.2363 29.6216 12.429 6.19441 15.102 10.8866 9.77283e+006
```

A profile on the R_0 parameter can also be run, using `spm -p > profile.log`. See the examples folder for an example of the output.

Simulations were run using command `spm -s 10 > simulations.log`. The first 20 lines of the first simulation are,

```
#[CAA-1998]
#report.type: simulated_observation
#observation.label: CAA-1998
@observation CAA-1998
age_plus_group True
categories immature + mature
delta 1e-11
layer cell
likelihood multinomial
max_age 30
min_age 1
process_error 50
selectivities FishingSel FishingSel
time_step one
tolerance 0.01
type proportions_at_age
year 1998
obs r2-c5 0 0 0 0 0.029411764705882353 0.058823529411764705 0.058823529411764705
    0.23529411764705882 0.20588235294117646 0.11764705882352941 0.14705882352941177
    0.058823529411764705 0.029411764705882353 0.058823529411764705 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0
obs r3-c5 0 0 0 0 0 0 0 0 0 0.027027027027027029 0.027027027027027029 0.027027027027027029
    0.10810810810810811 0.16216216216216217 0.13513513513513514 0.054054054054054057
    0.027027027027027029 0.027027027027027029 0 0.054054054054054057 0 0 0.054054054054054057
    0.027027027027027029 0.027027027027027029 0.027027027027027029 0.081081081081081086
    0.027027027027027029 0 0.10810810810810811
obs r3-c6 0.027027027027027029 0 0 0 0 0 0 0 0 0 0.027027027027027029 0.027027027027027029
    0.16216216216216217 0.081081081081081086 0.081081081081081086 0.054054054054054057
    0.10810810810810811 0 0.054054054054054057 0.081081081081081086 0.027027027027027029
    0.054054054054054057 0 0 0.054054054054054057 0 0 0.027027027027027029
    0.027027027027027029 0.10810810810810811
```

An estimate of the posterior distribution can be found using the command `spm -m -g 0 > MCMC.log`. The first set of output describes the covariance matrix and the MCMC proposal matrix. Following this, the log file contains the iterations, objective functions values, MCMC acceptance rate, and the step size.

The actual MCMC samples are listed at the end of the file.

14. Post processing output using R

The **R** package `spm` contains a set of **R** functions for reading `SPM` output, and is available as a precompiled binary for Microsoft Windows (.zip file) or as a source package (.gz file) for Linux. To check the version number and date of the `spm` **R** package (useful for checking that you have the most recent version), use the function `spm.version()`.

The `spm` **R** package includes a range of extract and write functions to aid post-processing of `SPM` input configuration files and output. The main extract functions are briefly described below. In addition, the package also some undocumented helper functions, that could be useful for writing you own analysis functions. See the the **R** help for more detail e.g., `help(spm)`

14.1. Read and extract reports from a SPM output file.

Command: `extract()`

Usage: `extract(file, path = "", ignore.unknown=FALSE)`

Arguments:

`file` the name of the SPM output file to read

`path` Optionally, the operating system path to the directory of the output file.

`ignore.unknown` Ignore unknown reports when reading. (This can be useful to read files that contain undocumented reports or other output)

Output: A list object with elements for each report type.

15. Troubleshooting

15.1. Introduction

SPM is a complex system, providing many opportunities for error — either because the parameter files do not correctly specify the model, or because the model specified does not work as expected. When in doubt, ask an experienced user. Debugging versions of SPM can also be compiled that help to track down cryptic errors.

When SPM generates an error and the error message makes no sense, please let the SPM authors know. Even if you manage to fix the problem yourself, we may be able to implement a more helpful error message and make life easier for the next person to encounter the problem. Guidelines for reporting an error are given in Section 15.3.

Some parameter values of functions or selectivities can result in either very large or very small numbers. These can, on occasion, generate internal numeric overflow errors within SPM. This is the most common cause of an overflow error, and can result in parameter estimates of NaN. The work-around to this type of error is to impose bounds on parameters that exclude the possibility of an overflow error.

15.2. Reporting errors

When reporting a bug or problem with SPM, please send a bug report to the authors. Use the text SPM: as the start of the subject line in the email. Note that following these guidelines will assist the SPM authors identify, reproduce, and hopefully solve any reported bugs.

Note that SPM is distributed as unsupported software. We are unable to provide much assistance to users — although we will usually endeavour to try. While we would appreciate being notified of any problems or errors in SPM, please note that we may not be able to provide timely solutions.

15.3. Guidelines for reporting a bug in SPM

1. Detail the version of SPM are you using? e.g., "SPM v

   Microsoft Windows executable"
2. What operating system or environment are you using? e.g., "IBM-PC Intel CPU running Microsoft Windows 7 Enterprise, Service Pack 1".
3. Give a brief one-line description of the problem, e.g., "a segmentation fault was reported".
4. If the problem is reproducible, please list the exact steps required to cause it, remembering to include the relevant SPM configuration file, other input files, and any out generated. Specify the *exact* command line arguments that were used, e.g., "Using the command `spm -e config.spm -q > logfile.out` reports a segmentation fault. The input configuration files are attached."
5. If the problem is not reproducible (only happened once, or occasionally for no apparent reason), please describe the circumstances in which it occurred and the symptoms observed (but note it is much harder to reproduce and hence fix non-reproducible bugs, but if several reports are made over time that relate to the same thing, then this may help to track down the problem), e.g., "SPM crashed, but I cannot reproduce how I did it. It seemed to be related to a local network crash but I cannot be sure."

6. If the problem causes any error messages to appear, please give the *exact* text displayed, e.g., `segmentation fault (core dumped)`.
7. Remember to attach all relevant input and output files so that the problem can be reproduced (it can helpful to compress these into a single file). Without these, it may not be possible to determine the cause of the problem. Note that it is helpful to be as specific as possible when describing the problem.

16. Acknowledgements

We thank Nokome Bentley (TROPHIA) and Ian Ball (Australian Antarctic Division) for their ideas that led to the development of the movement paradigm employed in this program. Thanks also to Adam Dunford, Andy McKenzie, Dave Gilbert, and Murray Smith (NIWA) for helpful discussions, suggestions for improvements, and error detection. The SPM logo was designed by Erika Mackay (NIWA). The copula preference function code and description in SPM were contributed by Craig Marsh and are described in Marsh (2015).

Much of the structure of SPM, equations, and documentation in this manual draw heavily on similar components of the fisheries population model CASAL (Bull et al., 2012). We thank the authors of CASAL for their permission to use their work as the basis for parts of SPM and allow the use of some of the definitions, concepts, and documentation from CASAL in SPM.

The development of SPM was funded by the New Zealand Ministry for Primary Industries, the Ministry of Business, Innovation & Employment (Project C01X1001, Protecting Ross Sea Ecosystems), and the National Institute of Water & Atmospheric Research Ltd. (NIWA) under NIWAs Fisheries Centre Research Programmes 1 and 3.

17. Quick reference

17.1. Population command and subcommand syntax

`@model` Define the spatial structure, population structure, annual cycle, and model years

`nrows` The number of rows n_{rows} in the spatial structure
`ncols` The number of columns n_{cols} in the spatial structure
`layer` The label for the base layer
`cell_length` The length (distance) of one side of a cell
`categories` Labels of the categories (rows) of the population component of the partition
`min_age` Minimum age of the population
`max_age` Maximum age of the population
`age_plus_group` Define the largest age as a plus group
`age_size` Define the label of the associated age-size relationship for each category
`initialisation_phases` Define the labels of the phases of the initialisation
`initial_year` Define the first year of the model, immediately following initialisation
`current_year` Define the current year of the model
`time_steps` Define the `@time_step` labels (in order that they are applied) to form the annual cycle

`@initialisation_phase` *`label`* Define the processes and years of the initialisation phase with label

`years` Define the number of years to run
`time_steps` Define the `@time_step` labels (in order that they are applied) in this initialisation phase
`lambda` Define the absolute proportional difference for assessing convergence between annual iterations during the initialisation
`lambda_years` Define the years to test for convergence during the initialisation

`@time_step` *`label`* Define a time-step with label

`processes` Define the process labels, in the order that they are applied, for the time-step

`@process` *`label`* Define a process with label

`type` Define the type of process

`@process[label].type=constant_recruitment`

`r0` Define the total amount of recruitment at equilibrium abundance levels
`categories` Define the categories into which recruitment occurs
`proportions` Define the proportion of recruitment that occurs into each category
`age` Define the age that receives recruitment
`layer` Name of the layer used to determine where recruitment occurs

`@process[label].type=bh_recruitment`

`r0` Define the total amount of recruitment at equilibrium abundance levels

`categories` Define the categories into which recruitment occurs
`proportions` Define the proportion of recruitment that occurs into each category
`age` Define the age that receives recruitment
`steepness` Define the Beverton-Holt stock recruitment relationship steepness (h) parameter
`b0` Define the `@initialisation_phase` label for the value of the derived quantity to use as the value of the spawning stock biomass (B_0)
`ssb` Define the label of the `@derived_quantity` that defines the spawning stock biomass (SSB)
`ssb_offset` Define the offset (in years) for the year of the derived quantity that is to be applied as the SSB in the stock-recruit relationship
`ycs_values` YCS values
`standardise_ycs_years` Years for which the year class strength values are defined to have mean 1.0
`layer` Name of the layer used to determine where recruitment occurs

`@process[label].type=local_bh_recruitment`

`r0` Define a multiplier of `r0_layer` for calculating the amount of recruitment in each cell at equilibrium abundance levels
`categories` Define the categories into which recruitment occurs
`proportions` Define the proportion of recruitment that occurs into each category
`age` Define the age that receives recruitment
`steepness` Define the Beverton-Holt stock recruitment relationship steepness (h) parameter
`b0` Define the `@initialisation_phase` label for the value of the derived quantity to use as the value of the spawning stock biomass (B_0) in each cell
`ssb` Define the label of the `@derived_quantity_by_cell` that defines the spawning stock biomass (SSB) in each cell
`ssb_offset` Define the offset (in years) for the year of the derived quantity by cell that is to be applied as the SSB in the stock-recruit relationship
`ycs_values` YCS values
`standardise_ycs_years` Years for which the year class strength values are defined to have mean 1.0
`layer` Define the label of the layer that defines the distribution of recruitment (as a multiplier of R_0 at equilibrium abundances in each cell

`@process[label].type=ageing`

`categories` Define the categories that ageing is applied to

`@process[label].type=constant_mortality_rate`

`m` Define the constant mortality rate to be applied
`categories` Define the categories that mortality is applied to
`selectivities` Define the selectivities applied to each category
`layer` Name of the layer

`@process[label].type=constant_exploitation_rate`

`u` Define the constant exploitation rate to be applied

u_max Define the maximum constant exploitation rate that could be applied
categories Define the categories that mortality is applied to
selectivities Define the selectivities applied to each category
layer Name of the layer

@process[label].type=annual_mortality_rate

years Define the years when the mortality rates are applied
m Define the mortality rate to be applied for each year
categories Define the categories that mortality is applied to
selectivities Define the selectivities applied to each category
layer Name of the multiplicative layer to be applied to *M*

@process[label].type=event_mortality

categories Define the categories that the event mortality is applied to
years Define the years where the mortality even is applied
layers Define the layers that specify the event mortality (as the abundance) in each year
u_max Define the maximum exploitation rate
selectivities Define the selectivities applied to each category
penalty Define the event mortality penalty label

@process[label].type=biomass_event_mortality

categories Define the categories that the event mortality is applied to
years Define the years where the mortality event is applied
layers Define the layers that specify the event mortality (as a biomass) in each year
u_max Define the maximum exploitation rate
selectivities Define the selectivities applied to each category
penalty Define the event mortality penalty label

@process[label].type=Holling_mortality_rate

is_abundance Is the mortality applied as a biomass or as abundance
a Define the *a* parameter of the Holling function
b Define the *b* parameter of the Holling function
x Define the type of Holling function or Michaelis-Menton function
categories Define the categories that the Holling mortality rate is applied to
selectivities Define the selectivities applied to each category
predator_categories Define the categories of the predator
predator_selectivities Define the selectivities applied to each predator category
u_max Define the maximum exploitation rate
penalty Define the event mortality penalty label

@process[label].type=Prey-suitability_predation

is_abundance Is the mortality applied as a biomass or as abundance
consumption_rate Define the total predator consumption rate
consumption_rate_layer Define the label of the layer that defines the predator consumption

rate in each cell
`prey_categories` Define the prey categories that the predation mortality is applied to
`prey_selectivities` Define the selectivities applied to each prey category
`electivities` Define the electivities applied to prey groups $1 \ldots n$
`predator_categories` Define the categories of the predator
`predator_selectivities` Define the selectivities applied to each predator category
`u_max` Define the maximum exploitation rate
`penalty` Define the process penalty label

`@process[label].type=category_state_by_age`

`category` Define the category that is the object of the process
`layer` Name of the categorical layer used to group the spatial cells for the process
`min_age` Define the minimum age for the process
`max_age` Define the maximum age for the process
`N [label]` Define the following data as the number of individuals to move in each age class

`@process[label].type=category_transition`

`from` Define the categories that are the source of the transition process
`selectivities` Define the selectivities applied to the source categories
`to` Define the categories that are the sink of the transition process
`years` Define the years where the category transition is applied
`layers` Define the layers that specify the transitions (as N for each cell) in each year
`u_max` Define the maximum proportion of individuals that can be moved
`penalty` Define the penalty to encourage models parameter values away from those which result in not enough individuals to move

`@process[label].type=category_transition_rate`

`from` Define the category that is the source of the transition process
`selectivities` Define the selectivities applied to the source categories
`to` Define the category that is the sink of the transition process
`proportions` Define the proportion of individuals to move
`layer` Name of the layer

`@process[label].type=category_transition_by_age`

`from` Define the categories that are the source of the transition process
`to` Define the categories that are the sink of the transition process
`year` Define the year when the category transition is applied
`layer` Name of the categorical layer used to group the spatial cells for the process
`min_age` Define the minimum age for the process
`max_age` Define the maximum age for the process
`N [label]` Define the following data as the number of individuals to move in each age class
`u_max` Define the maximum proportion of individuals that can be moved
`penalty` Define the penalty to encourage models parameter values away from those which result in not enough individuals to move

`@process[label].type=migration`

`categories` Define the categories that the migration movement event is applied to
`selectivities` Define the selectivities applied to each category
`proportion` Define the constant multiplier for the proportion of individuals that migrate
`source_layer` Define the label of a layer that defines the source cells of the migration movement event
`sink_layer` Define the label of a layer that defines the sink cells of the migration movement event

`@process[label].type=adjacent_cell`

`categories` Define the categories that the adjacent cell movement event is applied to
`selectivities` Define the selectivities applied to each category
`layer` Define the label of a gradient layer that defines the the relative strength of movement to adjacent cells
`proportion` Define the constant multiplier for the proportion that moves from each cell to the neighbouring cell

`@process[label].type=preference`

`categories` Define the categories that the preference function movement is applied to
`proportion` Define the constant multiplier for the proportion that the preference function movement is applied to
`preference_functions` Define the labels of the individual preference functions that make up the total preference function

`@process[label].type=preference_threaded`

`categories` Define the categories that the preference function movement is applied to
`proportion` Define the constant multiplier for the proportion that the preference function movement is applied to
`preference_functions` Define the labels of the individual preference functions that make up the total preference function

`@preference_function` *`label`* Define a preference function with label

`type` Define the type of preference function

`@preference_function[label].type=constant`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α

`@preference_function[label].type=normal`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`mu` Defines the μ parameter of the normal preference function
`sigma` Defines the σ parameter of the normal preference function

`@preference_function[label].type=double_normal`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`mu` Defines the μ parameter of the double-normal preference function
`sigma_l` Defines the σ_L parameter of the double-normal preference function
`sigma_r` Defines the σ_R parameter of the double-normal preference function

`@preference_function[label].type=logistic`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`a50` Defines the a_{50} parameter of the logistic preference function
`ato95` Defines the a_{to95} parameter of the logistic preference function

`@preference_function[label].type=inverse_logistic`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`a50` Defines the a_{50} parameter of the inverse-logistic preference function
`ato95` Defines the a_{to95} parameter of the inverse-logistic preference function

`@preference_function[label].type=exponential`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`lambda` Defines the λ parameter of the exponential preference function

`@preference_function[label].type=threshold`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`n` Defines the N parameter of the threshold preference function
`lambda` Defines the λ parameter of the threshold preference function

`@preference_function[label].type=categorical`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`category_labels` Defines the unique labels of `layer` in order of their coefficients
`category_values` Defines the coefficients for each unique label of `layer` in order of their labels

`@preference_function[label].type=monotonic_categorical`

`layer` Defines the layer which supplies the preference function independent variable
`alpha` Defines the multiplicative constant α
`category_labels` Defines the unique labels of `layer` in order of their coefficients
`category_values` Defines the coefficients for each unique label of `layer` in order of their labels

`@preference_function[label].type=gaussian_copula`

`rho` Defines the dependence parameter (ρ) for the copula

`layers` Defines the two layers which supplies the preference function independent variables
`pdf` Defines the two PDFs for the copula

`@preference_function[label].type=gumbel_copula`

`rho` Defines the dependence parameter (ρ) for the copula
`layers` Defines the two layers which supplies the preference function independent variables
`pdf` Defines the two PDFs for the copula

`@preference_function[label].type=frank_copula`

`rho` Defines the dependence parameter (ρ) for the copula
`layers` Defines the two layers which supplies the preference function independent variables
`pdf` Defines the two PDF for the copula

`@preference_function[label].type=independence_copula`

`layers` Defines the two layers which supplies the preference function independent variables
`pdf` Defines the two PDFs for the copula

`@pdf` *`label`* Define a PDF for use as a copula preference function with label

`type` Define the type of PDF

`@pdf[label].type=normal`

`mu` Define the mean of the PDF
`sigma` Define the variance of the PDF

`@pdf[label].type=lognormal`

`mu` Define the mean of the PDF
`sigma` Define the variance of the PDF

`@pdf[label].type=exponential`

`lambda` Define the mean of the PDF

`@pdf[label].type=uniform`

`a` Define the minimum of the PDF
`b` Define the maximum of the PDF

`@layer` *`label`* Define a layer function with label

`type` Define the type of layer

`@layer[label].type=numeric`

`data` Define the values of the layer
`rescale` If defined, then the values of the layer are rescaled to sum to this value

@layer[label].type=categorical

`data` Define the values of the layer

@layer[label].type=distance

@layer[label].type=haversine

`latitude` Define the layer that specifies the latitudes for each cell
`longitude` Define the layer that specifies the longitudes for each cell

@layer[label].type=dijkstra

@layer[label].type=haversine_dijkstra

`latitude` Define the layer that specifies the latitudes for each cell
`longitude` Define the layer that specifies the longitudes for each cell

@layer[label].type=abundance

`categories` Define the categories are used to calculate the abundance
`selectivities` Define the selectivities applied to each category

@layer[label].type=biomass

`categories` Define the categories are used to calculate the biomass
`selectivities` Define the selectivities applied to each category

@layer[label].type=abundance_density

`categories` Define the categories are used to calculate the abundance
`selectivities` Define the selectivities applied to each category

@layer[label].type=biomass_density

`categories` Define the categories are used to calculate the biomass
`selectivities` Define the selectivities applied to each category

@layer[label].type=numeric_meta

`default_layer` Define the default layer to use in years or initialisation phases where it is not otherwise defined
`years` Define the years that have a non-default layer
`layers` Define the layers for each of the years
`initialisation_phases` Define the initialisation phases that have a non-default layer
`initialisation_layers` Define the layers for each of the initialisation phases

@layer[label].type=categorical_meta

`default_layer` Define the default layer to use in years or initialisation phases where it is not

otherwise defined
`years` Define the years that have a non-default layer
`layers` Define the layers for each of the years
`initialisation_phases` Define the initialisation phases that have a non-default layer
`initialisation_layers` Define the layers for each of the initialisation phases

`@layer[label].type=derived_quantity`

`derived_quantity` Define the label of the `@derived_quantity` that is used as the source for the layer
`year_offset` Define the offset (in years) for the year of the derived quantity that is to be applied

`@layer[label].type=derived_quantity_by_cell`

`derived_quantity_by_cell` Define the label of the `@derived_quantity_by_cell` that is used as the source for the layer
`year_offset` Define the offset (in years) for the year of the derived quantity by cell that is to be applied

`@derived_quantity_by_cell` *`label`* Define a derived quantity by cell with label

`type` Define the type of derived quantity by cell

`@derived_quantity_by_cell[label].type=abundance`

`categories` Define the categories are used to calculate the derived quantity by cell
`selectivities` Define the selectivities
`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity by cell is calculated
`time_step` Define the time-step at the end of which the derived quantity by cell is calculated
`layer` Define the layer to be used in the calculations

`@derived_quantity_by_cell[label].type=biomass`

`categories` Define the categories are used to calculate the derived quantity by cell
`selectivities` Define the selectivities
`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity by cell is calculated
`time_step` Define the time-step at the end of which the derived quantity by cell is calculated
`layer` Define the layer to be used in the calculations

`@derived_quantity` *`label`* Define a derived quantity with label

`type` Define the type of derived quantity

`@derived_quantity[label].type=abundance`

`categories` Define the categories are used to calculate the derived quantity
`selectivities` Define the selectivities
`initialisation_time_steps` Define the time-steps during the initialisation phases at the end

of which the derived quantity is calculated
`time_step` Define the time-step at the end of which the derived quantity is calculated
`layer` Define the layer to be used in the calculations

`@derived_quantity[label].type=biomass`

`categories` Define the categories are used to calculate the derived quantity
`selectivities` Define the selectivities
`initialisation_time_steps` Define the time-steps during the initialisation phases at the end of which the derived quantity is calculated
`time_step` Define the time-step at the end of which the derived quantity is calculated
`layer` Define the layer to be used in the calculations

`@age_size` *`label`* Define an age-size relationship with label

`type` Define the type of size-at-age relationship

`@age_size[label].type=von_bertalanffy`

`linf` Define the L_∞ parameter of the von Bertalanffy relationship
`k` Define the k parameter of the von Bertalanffy relationship
`t0` Define the t_0 parameter of the von Bertalanffy relationship
`size_weight` Define the label of the associated size-weight relationship

`@age_size[label].type=schnute`

`y1` Define the y_1 parameter of the Schnute relationship
`y2` Define the y_2 parameter of the Schnute relationship
`tau1` Define the τ_1 parameter of the Schnute relationship
`tau2` Define the τ_2 parameter of the Schnute relationship
`a` Define the a parameter of the Schnute relationship
`b` Define the b parameter of the Schnute relationship
`size_weight` Define the label of the associated size-weight relationship

`@size_weight` *`label`* Define a size-weight relationship with label

`type` Define the type of relationship

`@size_weight[label].type=none`

`@size_weight[label].type=basic`

`a` Define the a parameter of the basic size-weight relationship
`b` Define the b parameter of the basic size-weight relationship

`@selectivity` *`label`* Define a selectivity function with label

`type` Define the type of selectivity function

`@selectivity[label].type=constant`

`c` Defines the C parameter of the selectivity function

@selectivity[label].type=knife_edge

e Defines the E parameter of the selectivity function

@selectivity[label].type=all_values

v Defines the V parameters (one for each age class) of the selectivity function

@selectivity[label].type=all_values_bounded

l Defines the L parameter of the selectivity function
h Defines the H parameter of the selectivity function
v Defines the V parameters (one for each age class from L to H) of the selectivity function

@selectivity[label].type=increasing

alpha Defines the α parameter of the selectivity function
l Defines the L parameter of the selectivity function
h Defines the H parameter of the selectivity function
v Defines the V parameters (one for each age class from L to H) of the selectivity function

@selectivity[label].type=logistic

alpha Defines the α parameter of the selectivity function
a50 Defines the a_{50} parameter of the selectivity function
ato95 Defines the a_{to95} parameter of the selectivity function

@selectivity[label].type=inverse_logistic

alpha Defines the α parameter of the selectivity function
a50 Defines the a_{50} parameter of the selectivity function
ato95 Defines the a_{to95} parameter of the selectivity function

@selectivity[label].type=logistic_producing

alpha Defines the α parameter of the selectivity function
l Defines the L parameter of the selectivity function
h Defines the H parameter of the selectivity function
a50 Defines the a_{50} parameter of the selectivity function
ato95 Defines the a_{to95} parameter of the selectivity function

@selectivity[label].type=double_normal

alpha Defines the α parameter of the selectivity function
mu Defines the μ parameter of the selectivity function
sigma_l Defines the σ_L parameter of the selectivity function
sigma_r Defines the σ_R parameter of the selectivity function

@selectivity[label].type=double_exponential

alpha Defines the α parameter of the selectivity function

`x1` Defines the x_1 parameter of the selectivity function
`x2` Defines the x_2 parameter of the selectivity function
`x0` Defines the x_0 parameter of the selectivity function
`y0` Defines the y_0 parameter of the selectivity function
`y1` Defines the y_1 parameter of the selectivity function
`y2` Defines the y_2 parameter of the selectivity function

`@selectivity[label].type=spline`

`alpha` Defines the α parameter of the selectivity function
`knots` Defines the locations of the knots for the cubic spline function
`values` Defines the values at the knots for the cubic spline function
`method` The method for constraining the end values of the spline

17.2. Estimation command and subcommand syntax

`@estimation`

`minimiser` The label of the minimiser to use, if doing a point estimate
`mcmc` The label of the MCMC to use, if doing an MCMC
`profile` The labels of the profiles to use, if doing a profile

`@minimiser` *`label`* Define the an minimiser estimator with label

`type` Define the type of minimiser

`@minimiser[label].type=numerical_differences`

`iterations` Define the maximum number of iterations for the minimiser
`evaluations` Define the maximum number of evaluations for the minimiser
`stepsize` Define the stepsize for the minimiser
`tolerance` Define the convergence criteria (tolerance) for the minimiser
`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating

`@minimiser[label].type=de_solver`

`population_size` Define the minimisers number of populations to generate
`crossover_probability` Define the minimisers crossover probability
`difference_scale` Define the scale of the difference of the parent candidates for the minimiser
`max_generations` Define the maximum generations for the minimiser convergence
`tolerance` Define the convergence criteria (tolerance) for the minimiser
`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating

`@mcmc` *`label`* Define the MCMC estimation arguments

`type` Define the method of MCMC

`@mcmc.type=metropolis_hastings`

`start` Covariance multiplier for the starting point of the Markov chain

`length` Length of the Markov chain
`keep` Spacing between recorded values in the chain
`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
`covariance_adjustment_method` Method for adjusting small variances in the covariance proposal matrix
`correlation_adjustment_diff` Minimum non-zero variance times the range of the bounds in the covariance matrix of the proposal distribution
`stepsize` Initial stepsize (as a multiplier of the approximate covariance matrix)
`proposal_distribution` The shape of the proposal distribution (either *t* or normal)
`df` Degrees of freedom of the multivariate t proposal distribution
`adapt_stepsize_at` Iterations in the chain to check and resize the MCMC stepsize

`@profile` *`label`* Define the profile parameters and arguments

`parameter` Name of the parameter to be profiled
`steps` Number of steps (values) at which to profile the parameter
`lower_bound` lower bound on parameter
`upper_bound` Upper bound on parameter

`@estimate` *`parameter_name`* Estimate an estimable parameter

`same` Names of the other parameters which are constrained to have the same value
`estimation_phase` Phase at which this parameter should be estimated, in point estimation
`lower_bound` Lower bounds on this parameter
`upper_bound` Upper bound on this parameter
`mcmc_fixed` Should this parameter be fixed during MCMC?
`type` Defines the type of prior for this parameter

`@estimate[label].type=uniform`

`@estimate[label].type=uniform_log`

`@estimate[label].type=normal`

`mu` Defines the mean μ of the normal prior
`cv` Defines the c.v. c of the normal prior

`@estimate[label].type=normal_by_stdev`

`mu` Defines the mean μ of the normal by standard deviation prior
`sigma` Defines the standard deviation σ of the normal by standard deviation prior

`@estimate[label].type=lognormal`

`mu` Defines the mean μ of the lognormal prior
`cv` Defines the c.v. c of the lognormal prior

`@estimate[label].type=beta`

`a` The lower value of the range parameter A of the Beta prior

`b` The upper value of the range parameter B of the Beta prior
`mu` Defines the mean μ of the Beta prior
`sigma` Defines the standard deviation σ of the Beta prior

`@catchability` *`label`* Define a catchability constant with `label`

`q` Value of the q parameter

`@penalty` *`label`* Define a penalty with `label`

`log_scale` Defines if the penalty in calculated in log space
`multiplier` Penalty multiplier

17.3. Observation command and subcommand syntax

`@observation` *`label`* Define an observation

`type` Define the type of observation

`@observation[label].type=proportions_at_age`

`year` Define the year that the observation applies to
`time_step` Define the time-step that the observation applies to
`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
`categories` Define the categories
`selectivities` Define the selectivities applied to each category
`min_age` Define the minimum age for the observation
`max_age` Define the maximum age for the observation
`age_plus_group` Define is the the maximum age for the observation is a plus group
`ageing_error` Define the label of the ageing-error matrix to be applied (if any)
`layer` Name of the categorical layer used to group the spatial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`tolerance` Define the tolerance on the sum-to-one error check in `SPM`
`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation

`@observation[label].type=proportions_by_category`

`year` Define the year that the observation applies to
`time_step` Define the time-step that the observation applies to
`proportion_method` Define the method for interpolating the time-step for calculating the

expected value of the observation
`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
`categories` Define the categories that make up the numerator of the observation
`categories2` Define the categories that, in combination with the numberator categories, make up the denominator
`selectivities` Define the selectivities applied to each category
`selectivities2` Define the selectivities applied to each category
`min_age` Define the minimum age for the observation
`max_age` Define the maximum age for the observation
`age_plus_group` Define is the the maximum age for the observation is a plus group
`ageing_error` Define the label of the ageing-error matrix to be applied
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`error_value [label]` Define the following data as error values (e.g., *N* for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation

`@observation[label].type=abundance`

`year` Define the year that the observation applies to
`time_step` Define the time-step that the observation applies to
`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
`catchability` Define the catchability constant label for the observation
`categories` Define the categories for which the observations occur
`selectivities` Define the selectivities applied to each category
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`error_value [label]` Define the following data as error values (e.g., *N* for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation

`@observation[label].type=biomass`

`year` Define the year that the observation applies to
`time_step` Define the time-step that the observation applies to
`proportion_method` Define the method for interpolating the time-step for calculating the

expected value of the observation
`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
`catchability` Define the catchability constant label for the observation
`categories` Define the categories for which the observations occur
`selectivities` Define the selectivities applied to each category
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation

`@observation[label].type=presence`

`year` Define the year that the observation applies to
`time_step` Define the time-step that the observation applies to
`proportion_method` Define the method for interpolating the time-step for calculating the expected value of the observation
`proportion_time_step` Define the interpolated proportion through the time-step for calculating the expected value of the observation
`catchability` Define the catchability constant label for the observation
`categories` Define the categories for which the observations occur
`selectivities` Define the selectivities applied to each category
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`error_value [label]` Define the following data as error values (e.g., N for binomial likelihoods) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`simulation_likelihood` Define the likelihood when doing simulations, if the observations is a pseudo-observation

`@ageing_error` *`label`* Define ageing error with `label`

`type` The type of ageing error

`@ageing_error[label].type=none`

`@ageing_error[label].type=normal`

`cv` Parameter of the normal ageing error model
`k` The k parameter of the normal ageing error model

`@ageing_error[label].type=off_by_one`

p1 The p_1 parameter of the off-by-one ageing error model
p2 The p_2 parameter of the off-by-one ageing error model
k The k parameter of the off-by-one ageing error model

17.4. Report command and subcommand syntax

@report *label* Define an output report

type Define the type of report

@report[label].type=spatial_map

file_name Define the name of the output file where the report is written

@report[label].type=partition

years Define the years that the partition report applies to
time_step Define the time-step that the partition report applies to
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=partition_biomass

years Define the years that the partition_biomass report applies to
time_step Define the time-step that the partition_biomass report applies to
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=initialisation

initialisation_phase Define the phase of initialisation that the partition report applies to
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=process

process Define the label of the process to summarise
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=preference_function

preference_function Define the label of the preference function to summarise
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=derived_quantity

derived_quantity Define the label of the derived quantity to print
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=derived_quantity_by_cell

derived_quantity_by_cell Define the label of the derived quantity by cell to print
initialisation Specify if the derived quantity by cell values for each year of the initialisation phases should be also be output
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=estimate_summary

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=estimate_value

file_name Define the name of the output file where the report is written
header Specify if the output contains the standard SPM style header at the start of the output
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=objective_function

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=covariance

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=observation

observation Define the label of the observation to print
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=simulated_observation

observation Define the label of the observation from which to simulate values

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=ageing_error

ageing_error Define the label of the ageing_error misclassification matrix
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=layer

layer Define the label of the layer to print
years Define the years for the printing of the layer
time_step Define the time-step for the printing of the layer
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=layer_derived_view

layer Define the label of the layer to print
years Define the years for the printing of the layer
time_step Define the time-step for the printing of the layer
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=selectivity

selectivity Define the label of the selectivity to print
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=random_number_seed

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=age_size

age_size Define the label of the age-size relationship
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=size_weight

`age_size` Define the label of the age-size command that specifies the size-weight relationship
`sizes` Define the sizes for which to report the mean weights
`file_name` Define the name of the output file where the report is written
`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to

`@report[label].type=age_weight`

`age_size` Define the label of the age-size relationship
`file_name` Define the name of the output file where the report is written
`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to

`@report[label].type=MCMC`

`file_name` Define the name of the output file where the report is written
`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to

`@report[label].type=MCMC_samples`

`file_name` Define the name of the output file where the report is written
`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to

`@report[label].type=MCMC_objectives`

`file_name` Define the name of the output file where the report is written
`overwrite` Specify if any previous file with the same name as the output file should be overwritten or appended to

17.5. Other commands and subcommands

`@include` *`file`* Include an external file

18. References

I. Ball and A Constable. Fish heaven: A Monte Carlo, spatially explicit single species fishery model for the testing of parameter estimation methods. Technical Report WG-FSA-00/36, Australian Antarctic Division, October 2000.

I. Ball and A.T. Williamson. Fish heaven user's manual. Technical report, Australian Antarctic Division, August 2003.

N. Bentley, C.R. Davies, S.E. McNeill, and N.M. Davies. A framework for evaluating spatial closures as a fisheries management tool. New Zealand Fisheries Assessment Report, Ministry of Fisheries, 2004a.

N. Bentley, N.M. Davies, and S.E. McNeill. A spatially explicit model of the snapper (*Pagrus auratus*) fishery in SNA1. New Zealand Fisheries Assessment Report, Ministry of Fisheries, 2004b.

R.J.H. Beverton and S.J. Holt. *On the dynamics of exploited fish populations*. Fishery investigations. HMSO, London, 1957.

B. Bull, R.I.C.C. Francis, A. Dunn, A. McKenzie, D.J. Gilbert, M.H. Smith, R. Bian, and D. Fu. CASAL C++ Algorithmic Stock Assessment Laboratory): CASAL user manual v2.30-2012/03/21. Technical Report 135, NIWA, 2012.

J. E. Dennis Jr and R.B. Schnabel. *Numerical methods for unconstrained optimisation and nonlinear equations*. Classics in Applied Mathematics. Prentice Hall, 1996.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271, 1959.

A B Gelman, J S Carlin, H S Stern, and D B Rubin. *Bayesian data analysis*. Chapman and Hall, London, 1995.

W R Gilks, A Thomas, and D J Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

C. S. Holling. The components of predation as revealed by a study of small-mammal predation of the european pine sawfly. *The Canadian Entomologist*, 91:293–320, 1959.

J. Jurado-Molina, P.A. Livingston, and J.N. Ianelli. Incorporating predation interactions in a statistical catch-at-age model for a predatorprey system in the eastern bering sea. *Canadian Journal of Fisheries and Aquatic Sciences*, 62:1865–1873, 2005.

C. Marsh. Modelling spatio-temporal preferences for albacore tuna. Master's thesis, Victoria University of Wellington, New Zealand, 2015.

Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation: Special Issue on Uniform Random Number Generation*, 8(1):3–30, January 1998.

M.I. Michaelis and L. Menten. Die kinetik der invertinwirkung. *Biochemische Zeitschrift*, 49: 333369, 1913.

Andre E. Punt and Ray Hilborn. *BAYES-SA. Bayesian stock assessment methods in fisheries. User's manual. FAO Computerized information series (fisheries) 12*. Food and Agriculture Organisation of the United Nations, Rome (Italy)., 2001.

J Schnute. A versatile growth model with statistically stable parameters. *Canadian Journal of Fisheries and Aquatic Sciences*, 38(9):1128–1140, 1981.

Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995. URL `http://citeseer.ist.psu.edu/182432.html`.

19. Spatial Population Model software license

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE (”AGREEMENT”). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT’S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

”Contribution” means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution ’originates’ from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor’s behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

”Contributor” means any person or entity that distributes the Program.

”Licensed Patents ” mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

”Program” means the Contributions distributed in accordance with this Agreement.

”Recipient” means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients

to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

20. Index