

# Spatial Population Model User Manual

SPM v1.00-2009-03-17 (rev. 3056)

Alistair Dunn, Scott Rasmussen

March 2009



Spatial Population Model User Manual (modified: 2009-03-16)  
for use with SPM v1.00-2009-03-17 (rev. 3056)



Citation:

Dunn, A.; Rasmussen, S. (2009) Spatial Population Model User Manual, SPM v1.00-2009-03-17 (rev. 3056) . National Institute of Water & Atmospheric Research Ltd. *Unpublished report*. 134 p.



## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Version . . . . .	1
1.2	Citing SPM . . . . .	1
1.3	Software license . . . . .	1
1.4	System requirements . . . . .	1
1.5	Necessary files . . . . .	2
1.6	Useful add-ons . . . . .	2
1.7	Getting help . . . . .	2
1.8	Technical details . . . . .	2
<b>2</b>	<b>Model overview</b>	<b>3</b>
2.1	Introduction . . . . .	3
2.2	The population section . . . . .	4
2.3	The estimation section . . . . .	4
2.4	The observation section . . . . .	4
2.5	The report section . . . . .	5
<b>3</b>	<b>Running SPM</b>	<b>7</b>
3.1	Using SPM . . . . .	7
3.2	The input configuration file . . . . .	7
3.3	Redirecting standard output . . . . .	7
3.4	Command line arguments . . . . .	8
3.5	Constructing an SPM input configuration file . . . . .	9
3.5.1	Commands . . . . .	9
3.5.2	Subcommands . . . . .	10
3.5.3	The command-block format . . . . .	10
3.5.4	Commenting out lines . . . . .	11
3.5.5	Determining parameter names . . . . .	11
3.6	SPM exit status values . . . . .	12
<b>4</b>	<b>The population section</b>	<b>13</b>
4.1	Introduction . . . . .	13
4.2	Spatial structure . . . . .	13
4.3	Population structure . . . . .	14
4.4	Layers . . . . .	15
4.5	Time sequences . . . . .	17
4.5.1	Annual cycle . . . . .	18
4.5.2	Model initialisation . . . . .	18
4.5.3	Model years . . . . .	18
4.5.4	Projections . . . . .	19
4.6	Processes . . . . .	19

4.7	Population processes . . . . .	20
4.7.1	Recruitment . . . . .	20
4.7.2	Ageing . . . . .	20
4.7.3	Mortality . . . . .	21
4.7.4	Category transitions . . . . .	22
4.8	Movement processes . . . . .	22
4.8.1	Migration movement . . . . .	23
4.8.2	Adjacent cell movement . . . . .	23
4.8.3	Preference movement . . . . .	23
4.9	Derived quantities . . . . .	25
4.10	Size-at-age . . . . .	25
4.11	Mean weight . . . . .	26
4.12	Selectivities . . . . .	26
4.12.1	Constant . . . . .	27
4.12.2	Knife-edge . . . . .	27
4.12.3	All-values . . . . .	27
4.12.4	All-values-bounded . . . . .	27
4.12.5	Increasing . . . . .	27
4.12.6	Logistic . . . . .	28
4.12.7	Logistic producing . . . . .	28
4.12.8	Double-normal . . . . .	28
4.12.9	Double-exponential . . . . .	28
<b>5</b>	<b>The estimation section</b>	<b>29</b>
5.1	Role of the estimation section . . . . .	29
5.2	Specifying the parameters to be estimated . . . . .	29
5.3	Point estimation . . . . .	29
5.4	Posterior profiles . . . . .	30
5.5	Bayesian estimation . . . . .	30
<b>6</b>	<b>The observation section</b>	<b>35</b>
6.1	Types of observations . . . . .	35
6.1.1	Event mortality proportions-at-age . . . . .	35
6.1.2	Proportions-at-age . . . . .	35
6.1.3	Proportions-by-category . . . . .	35
6.1.4	Abundance . . . . .	35
6.2	The objective function . . . . .	36
6.2.1	The objective function . . . . .	36
6.2.2	Likelihoods . . . . .	36
6.2.3	Priors . . . . .	36
6.2.4	Penalties . . . . .	36
6.3	Simulating observations . . . . .	36

6.4	Pseudo-observations . . . . .	36
<b>7</b>	<b>The report section</b>	<b>37</b>
7.1	Reporting model outputs . . . . .	37
7.1.1	Printing the partition . . . . .	37
7.1.2	Printing derived quantities . . . . .	37
7.1.3	Printing the objective function . . . . .	37
7.1.4	Printing observations, fits, and residuals . . . . .	37
7.1.5	Printing layers . . . . .	37
7.1.6	Printing selectivities . . . . .	37
7.1.7	Verifying the size-weight relationship . . . . .	37
<b>8</b>	<b>Population command and subcommand syntax</b>	<b>39</b>
8.1	Model structure . . . . .	39
8.2	Initialisation . . . . .	40
8.2.1	Iterative initialisation . . . . .	41
8.3	Time steps . . . . .	41
8.4	Processes . . . . .	41
8.4.1	Constant recruitment process . . . . .	42
8.4.2	Beverton-Holt recruitment process . . . . .	42
8.4.3	Ageing process . . . . .	44
8.4.4	Constant mortality rate process . . . . .	44
8.4.5	Annual mortality rate process . . . . .	45
8.4.6	Event mortality process . . . . .	45
8.4.7	Biomass event mortality process . . . . .	46
8.4.8	Category transition process . . . . .	47
8.4.9	Category transition rate process . . . . .	48
8.4.10	Migration movement . . . . .	48
8.4.11	Adjacent cell movement . . . . .	49
8.4.12	Preference movement . . . . .	49
8.5	Preference functions . . . . .	49
8.5.1	Constant . . . . .	50
8.5.2	Normal . . . . .	50
8.5.3	Double-normal . . . . .	51
8.5.4	Logistic . . . . .	51
8.5.5	Inverse-logistic . . . . .	52
8.5.6	Exponential . . . . .	52
8.5.7	Threshold . . . . .	52
8.5.8	Threshold-biomass . . . . .	53
8.6	Layers . . . . .	54
8.6.1	Numeric . . . . .	54
8.6.2	Categorical . . . . .	54

8.6.3	Distance . . . . .	55
8.6.4	Abundance . . . . .	55
8.6.5	Biomass . . . . .	55
8.6.6	Abundance-density . . . . .	55
8.6.7	Biomass-density . . . . .	56
8.6.8	Meta-layer . . . . .	56
8.7	Derived quantities . . . . .	56
8.7.1	Abundance . . . . .	57
8.7.2	Biomass . . . . .	57
8.8	Size-at-age . . . . .	58
8.8.1	von Bertalanffy . . . . .	58
8.8.2	Schnute . . . . .	59
8.9	Size-weight . . . . .	60
8.9.1	None . . . . .	60
8.9.2	Basic . . . . .	61
8.10	Selectivities . . . . .	61
8.10.1	Constant . . . . .	61
8.10.2	Knife-edge . . . . .	62
8.10.3	All-values . . . . .	62
8.10.4	All-values-bounded . . . . .	62
8.10.5	Increasing . . . . .	62
8.10.6	Logistic . . . . .	63
8.10.7	Logistic producing . . . . .	63
8.10.8	Double-normal . . . . .	64
8.10.9	Double-exponential . . . . .	64
8.11	Joint selectivities . . . . .	65
<b>9</b>	<b>Estimation command and subcommand syntax</b>	<b>67</b>
9.1	Estimation methods . . . . .	67
9.2	Point estimation . . . . .	67
9.2.1	Numerical differences minimiser . . . . .	68
9.2.2	Differential evolution minimiser . . . . .	68
9.3	Monte Carlo Markov Chain (MCMC) . . . . .	69
9.3.1	Metropolis-Hastings . . . . .	69
9.4	Profiles . . . . .	70
9.5	Defining the parameters to be estimated and their priors . . . . .	71
9.6	Priors . . . . .	72
9.6.1	Uniform prior . . . . .	72
9.6.2	Uniform-log prior . . . . .	72
9.6.3	Normal prior . . . . .	72
9.6.4	Normal-by-sd prior . . . . .	73
9.6.5	Lognormal prior . . . . .	73



9.6.6	Beta prior . . . . .	73
9.7	Defining ageing error . . . . .	74
9.7.1	No ageing error . . . . .	74
9.7.2	Normal ageing error . . . . .	74
9.7.3	Off-by-one ageing error . . . . .	74
9.8	Defining catchability constants . . . . .	75
9.9	Defining penalties . . . . .	75
<b>10</b>	<b>Observation command and subcommand syntax</b>	<b>77</b>
10.1	Observation types . . . . .	77
10.1.1	Event mortality-at-age . . . . .	77
10.1.2	Proportions at age . . . . .	79
10.1.3	Proportions by category . . . . .	81
10.1.4	Abundance . . . . .	83
10.1.5	Biomass . . . . .	84
<b>11</b>	<b>Reporting command and subcommand syntax</b>	<b>87</b>
11.1	Reports . . . . .	87
11.1.1	Print the partition . . . . .	87
11.1.2	Print the partition at initialisation . . . . .	88
11.1.3	Print a derived quantity . . . . .	88
11.1.4	Printing the estimated parameter values out as a vector . . . . .	88
11.1.5	Print a summary of the estimated parameters . . . . .	89
11.1.6	Print the Objective function values . . . . .	89
11.1.7	Print a a summary of the an observation, including fits, and residuals . . . . .	89
11.1.8	Print a layer . . . . .	90
11.1.9	Print a selectivity . . . . .	90
11.1.10	Print the random number seed used . . . . .	91
11.1.11	Print the weight-at-size . . . . .	91
<b>12</b>	<b>Other commands and subcommands</b>	<b>93</b>
<b>13</b>	<b>Examples</b>	<b>95</b>
13.1	An example of a simple $1 \times 1$ non-spatial model . . . . .	95
13.2	An example of a simple $10 \times 10$ spatial model . . . . .	95
13.3	An example of a more complex $10 \times 10$ spatial model . . . . .	96
<b>14</b>	<b>Post processing output using R</b>	<b>97</b>
<b>15</b>	<b>Troubleshooting</b>	<b>99</b>
15.1	Introduction . . . . .	99
15.2	Reporting errors . . . . .	99
15.3	Guidelines for reporting a bug in SPM . . . . .	99

<b>16 Acknowledgements</b>	<b>101</b>
<b>17 Quick reference</b>	<b>103</b>
17.1 Population command and subcommand syntax . . . . .	103
17.2 Estimation command and subcommand syntax . . . . .	110
17.3 Observation command and subcommand syntax . . . . .	112
17.4 Reporting command and subcommand syntax . . . . .	114
17.5 Other commands and subcommands . . . . .	116
<b>18 References</b>	<b>117</b>
<b>19 Spatial Population Model software license</b>	<b>119</b>
<b>20 Index</b>	<b>123</b>

---

## 1 Introduction

The Spatial Population Model (SPM) 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. Movement can be modelled as either adjacent cell movements or global movements based on covariates.

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. If you are new to SPM, then a good place to start is by reading this manual and attempting to replicate the examples (Section 13).

### 1.1 Version

This document (last modified 2009-03-16) describes SPM v1.00-2009-03-17 (rev. 3056) . 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. (2009) Spatial Population Model User Manual, SPM v1.00-2009-03-17 (rev. 3056) . National Institute of Water & Atmospheric Research Ltd. *Unpublished report*. 134 p.

### 1.3 Software license

This program and the accompanying materials are made available under the terms of the Common Public License v1.0 which accompanies this distribution (Section 19).

Copyright ©2008-2009, National Institute of Water & Atmospheric Research Ltd. and the New Zealand Ministry of Fisheries. All rights reserved.

### 1.4 System requirements

SPM is available for most IBM compatible machines running Linux and from the command prompt under most Microsoft Windows operating systems.

Several of SPMs tasks are highly computer intensive and a fast, powerful processor is recommended. We recommend a minimum of 10 megabytes of free RAM (although, depending on the scope of the problem, you may need much more). Some of SPMs tasks can be multi-threaded, and hence multi-core machines may perform some tasks considerably 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. However, we note that, depending on the model implemented, some of SPMs tasks can take a considerable amount of time.

### 1.5 Necessary files

In Linux, only the executable file `spm` is required to run SPM (but, depending on your system, you may need the either the 32- or 64-bit version). For Microsoft Windows, you need the executable file `spm.exe`. There is no 64-bit version for Microsoft Windows.

### 1.6 Useful add-ons

No software other than the appropriate operating system or emulation package is required to run SPM. However, as SPM offers little in the way of post-processing of the output, most users will wish to have a package available that allows tabulation and graphing of model outputs. We recommend the use of software packages such as Microsoft Excel, S-Plus, or **R** (R Development Core Team 2007). See Section 14 for details of the `spm` **R** package for extracting SPM output.

### 1.7 Getting help

SPM is distributed as unsupported software. The authors do not, as a rule, provide help for users of SPM. However, we may be able to offer limited assistance, and 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.

### 1.8 Technical details

SPM is compiled on Linux using `gcc`, the C/C++ compiler developed by the GNU Project. The 32-bit Linux version has been compiled using `gcc` version 4.1.2 20070115 (prerelease) (SUSE Linux), the 64-bit Linux version uses `gcc` version 4.1.0 (SUSE Linux). Note that SPM is not supported for Linux kernel versions prior to 2.6. The Microsoft Windows version is compiled using Mingw32 `gcc` 3.4.5, and should run on most 32-bit WindowsXP and Windows Vista systems. There are no plans to port SPM to Microsoft Windows 64-bit platforms.

SPM uses two minimisers — the first is closely based on the main algorithm of Dennis Jr and Schnabel (1996), and 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 1.38.0).

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 on request.

---

## 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 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.

The basic structure of the model is defined in terms of the *state*. The state consists of two parts, the *partition* and *derived quantities*. 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 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. Hence, derived quantities build up a vector of values over the model run years. For example, the numbers of individuals in a category labelled mature at some point in the annual cycle may be a derived quantity. Hence, the state is the combination of the partition and the derived quantities at some instance in time. Changes to the partition occur by the application of *processes*. Additions to the vectors of derived quantities occur when the model is requested to add a value to each derived quantity vector.

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. Further, 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 they occur in the order that they are specified. Observations are always evaluated at the end of a time step. 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 by either adjacent cell movements, between cell migrations, or by global movements parametrised 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., 2008). 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)

The spatial component of SPM is designed to allow parametrise movement cohorts and groups of individuals between spatial locations, and hence allow movement parameters to be estimated by

incorporating spatially explicit observations.

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 command prompt window in Microsoft Windows or from a text terminal in Linux. A model can be either *run*, estimable parameters can be *estimated* or *profiled*, MCMC distributions calculated, and these estimates can be *projected* into the future or used as an operating model to *simulate* observations.

This section gives a quick overview of the model, and how to use it. Detailed descriptions of the components of SPM, the model structures, mathematical equations used, and command and subcommand arguments are given in the following sections.

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). These input configuration file completely describe 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), defines the 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 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.

Further, 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 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, the catch-limit penalties can be used to discourage parameter estimates that lead to models where the recorded catch 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. 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 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 command line in Microsoft Windows or from a terminal window in 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, define the parameters for a projection, or 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 command line 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). 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`. See your operating system documentation for help on identifying or modifying your `PATH`.

#### 3.2 The input configuration file

The input configuration file comprises of four components; 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 made up of a number of commands (many with subcommands) which specify various options for each of these components.

The command and subcommand definitions in the input configuration file can be extensive, 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 out to display run-time information. Standard error is not used by SPM, but may be used by the operating system to report an error with SPM. We suggest redirecting both the

standard out 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 if 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., the Professional version of WindowsXP), 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`.
- f** Project the model *forward* in time using the parameter values in the input configuration file as the starting point for the estimation, or optionally, with the start values from the file denoted with the command line argument `-i file`.
- s** *Simulate* observations 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 estimated parameter values from *file*. See Section 11.1.4 for details about the format of *file*.
- t number** Number of *threads* to run (i.e., number of processors available for use).
- 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 looks the command `@Estimation.random_seed` for a random number seed, and if not defined, then automatically generates 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.

In general, the input configuration file uses a command-block format to describe commands, subcommands, and their arguments and values. The input configuration file consists of any number of command-blocks in any order, but these must be within the [population], [estimation], and [report] sections for the population, estimation, and report commands respectively.

#### 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`)
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, but must not contain white-space or a full-point ('.').

### 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 8.11, and are summarised below.

Like commands (`@command`), subcommands and their arguments are not order specific — except 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

**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, 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. Commands and subcommands consist of letters and/or underscores, must not contain a spaces or full-point ('.').

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.

In general, commands, sub-commands, and arguments in the parameter files are case insensitive. But note, however, 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).

### 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.  
command[label].subcommand[i-j] if the command has a label, and the subcommand arguments are a vector, and we are accessing the elements from *i* to *j* (inclusive) 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. Hence, to determine if SPM completed its task successfully, check the standard output for error and information messages.

---

## 4 The population section

### 4.1 Introduction

The population section defines the models 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, at any point in time, is described by the *state*, which comprises of the *partition* and any *derived quantities*. The partition is a 4-dimensional matrix, comprising of the spatial structure, and within each spatial location, the population structure.

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 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, or shift numbers between ages and categories)
  - 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 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  $i = 1 \dots n_{rows}$  and columns  $j = 1 \dots n_{cols}$ . 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  $k = 1 \dots n_{categories}$  and ages  $l = 1 \dots n_{ages} = age_{min} \dots age_{max}$ ). 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*.

The spatial structure of SPM is represented by an  $n \times m$  grid, with rows  $i = 1 \dots n$  and columns  $j = 1 \dots m$ . Each cell of this matrix records the population structure at that point in space and is represented by an  $k \times l$  rectangular matrix (with categories  $k = 1 \dots k$  and ages  $l = age_{min} \dots age_{max}$ ). Hence we can describe any spatial and population element of the model as  $element(i, j, k, l)$ .

SPM implements two spatial structures, one derived from a grid of *square* cells — the default (Figure 4.1) and the other derived from a grid of north-south orientated *hexagons* (Figure 4.2). We note that 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  $1 \times 1$  grid (i.e., a single spatial cell). Associated with the 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) as values within the base layer that are 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, as used for density calculations within SPM.

	Col 1	Col 2	Col 3	Col 4
Row 1	(1,1)	(1,2)	(1,3)	(1,4)
Row 2	(2,1)	(2,2)	(2,3)	(2,4)
Row 3	(3,1)	(3,2)	(3,3)	(3,4)

**Figure 4.1: An illustration of the *square* spatial structure**

Models can be implemented as a grid based on either squares or hexagons, but not both. This choice effects the location of neighbours (for adjacent movements) and the distance between cells (for preference based movements). Distance between cells is determined as the euclidean distance between cell centres, modified by an arbitrary scalar.

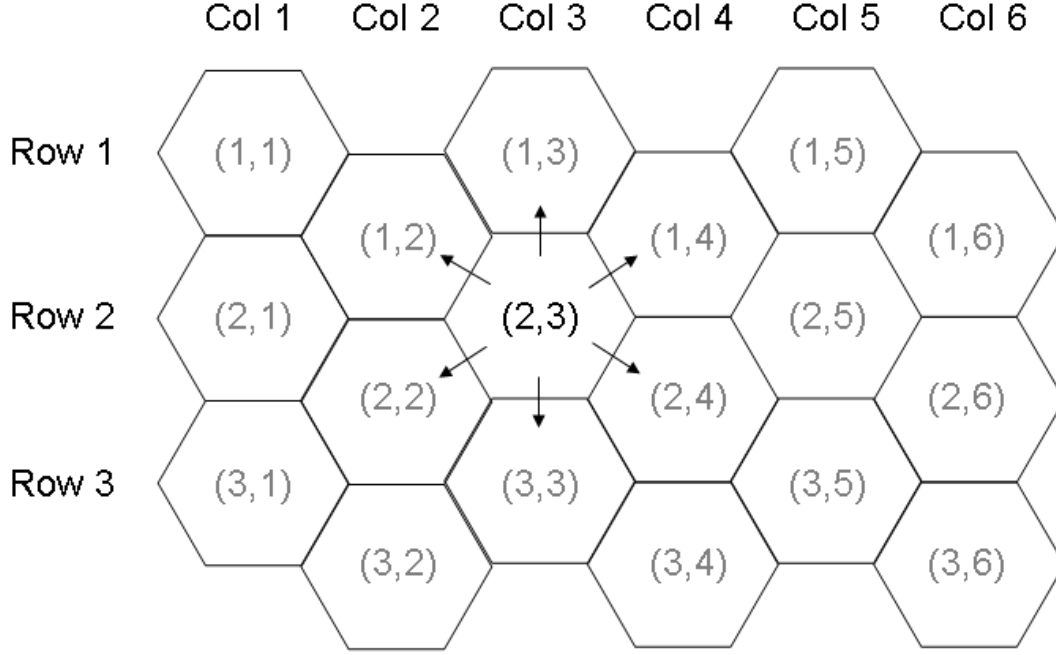
Hence, the definition of the spatial structure includes;

- The type of spatial grid and its dimensions,  $n_{rows}$  and  $m_{cols}$
- 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 (either a square or hexagon) to be used as the scaler for distance calculations

### 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  $k = 1 \dots n_{categories}$  and ages  $l = age_{min} \dots age_{max}$ .





**Figure 4.2: An illustration of the *hexagon* spatial structure**

Within each  $(i, j)$  element of the spatial grid element (termed spatial *cell*), 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  $k = 1 \dots n_{categories}$  and ages  $l = age_{min} \dots age_{max}$ .

The names and number of categories are user defined, but there just be at least one category in any model. The ages are defined as a sequence from  $age_{min}$  to  $age_{max}$ , and the last age may optionally be a plus group.

Hence, the definition of the population structure includes;

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

#### 4.4 Layers

*Layers* are used by SPM to evaluate locations where the population may be present (via the *base layer*), to provide sets of known attributes of each spatial location (for preference based movements), and to group or categorise cells for use by processes and observations. Layers consist of an  $n \times m$  matrix and can be either *logical*, *numeric*, or *categorical*. See Section 4 for further details.

Layers form 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.

Every model must define at least one layer, the base later  $L_B$ . A layer is defined as a  $n_{rows} \times n_{cols}$

grid of values (with one exception — the distance layer, see below), where the value for each cell represents a known quantity. For example layers may represent classifications, physical attributes, or some other assumed quantity. Typically they are provided by the user as a matrix of values, although abundance and distance layers can be calculated by SPM as and when required.

Within SPM, layers are used in three 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.
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 must be continuous (i.e., numeric) variables. Covariate layers must have values  $\geq 0$ .
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.

Typically, layers are supplied by the user and are assumed known and constant. 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.
2. Categorical layers: A model may have many categorical layers, and these are used as a classification or grouping variable for aggregating data over individual cells, e.g., management areas. Such layers are typically used to aggregate the population within cells into groups for comparing with observations. 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.
3. Distance layers: A distance layer is one that defines the distance between any two cells. By default, SPM calculates the values of the distance layer as the Euclidean distance (where the grid type is square). Here, the distance between cell  $a$  and cell  $b$  can be defined as,

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

where  $x$  and  $y$  represent the  $x$ - and  $y$ -coordinates of  $a$  and  $b$  respectively, and  $\lambda$  is an arbitrary scalar representing the length of one side of the square. 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 under this definition, the distance between any cell and itself is 0.

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  $l$ .

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

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

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  $w_{kl}$

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

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

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  $l$ ,

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

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

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  $w_{kl}$ ,

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

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

8. Meta-layers: In addition to the above types of layer, SPM defines a special type of layer known as a *meta-layer*. The meta-layer allows individual layers (of the same type) to be indexed by year, and applied as a single layer within the model. For example, assume that we had a model where we wished to use Sea Surface Temperature (SST) as a layer, perhaps to control some movement process. The SST values for each year of the model would be defined as individual numeric layers, each with a unique label. We could then define a meta-layer that indexed the individual annual SST layers by year, and use the meta-layer as the control layer in the movement process.

However, there are exceptions to this rule — layers of type biomass, quantity, and distance are calculated automatically by SPM as required. For example, for the distance layer in a square grid, the distance between cell  $a$  and cell  $b$  is defined as proportional to  $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ , where  $x$  and  $y$  represent the x- and y-coordinates of  $a$  and  $b$  respectively.

For the abundance layer, the abundance or biomass of a cell is simply a count of the number (or biomass) of individuals in the cell  $a$  within categories  $K$ , with selectivity  $S$ , e.g.,  $N(a) = \sum_k \sum_i \text{element}(i, j, k, l)$ .

Note that SPM does not ‘edit’ or otherwise change layers, including adding or otherwise combining layers that are supplied in the input parameter files.

## 4.5 Time sequences

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

- Initialisation
- Run years
- Projection 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. User-defined 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 are not implemented during the initialisation phases, and that the annual cycle in the initialisation phases can be different from that run during the model years.

### 4.5.2 Model initialisation

SPM initialises the initial equilibrium state as an iterative process, because a general solution that initialises complex structured movement models can be difficult to implement using analytic techniques. However, initialising via iteration for a long-lived species with complex movements can also 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.

The initialisation part can consist of one or more phases, with each phase occurring for at least one year. Within each phase, the processes defined for that phase are carried out, and use 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) seeded with a 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.

Hence, you need to define;

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

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. Analytical initialisation is not implemented in SPM, hence equilibrium and initial population states must be evaluated iteratively. At the end of the initialisation, 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, project forward to determine future states, or simulate observations from the current state.

### 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, and observations can be associated with the state of the model at the end of any time step, i.e., likelihoods for particular observations are evaluated, if required, at the end of each time step.

Processes are carried out in the order specified within each time step, and can be the same or different to processes in other initialisation phases of the model. The run years define the years over which the model is to run and the annual cycle within each year. The model runs from the start of year `initial` and runs to the end of year `current`. The projection part then extends the run time up to the end of year `final`.

- The time steps and the processes applied in each

- The initial year (i.e., the model start year)
- The current year (i.e., the model end year)
- The final year (i.e., the model projection end year)

#### 4.5.4 Projections

To be added...

### 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 movements.

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., 2008).
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.

### 4.7.1 Recruitment

Recruitment processes are defined as process that introduces new individuals into the model. SPM implements two types of recruitment process, constant recruitment and Beverton-Holt recruitment (Beverton and Holt, 1957).

In both of the recruitment process, a number of individuals are added to the partition at the age and categories specified. If more than one category is defined, then the proportions of individuals added across categories are user-defined. 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.

For each cell where cell( $i, j$ ) is a member of some layer  $L_R$ , the number of fish added in year  $y$  is

$$\text{element}(i, j, k, l) \leftarrow \text{element}(i, j, k, l) + p_k(R_y/n) \quad (4.6)$$

where age is the age defined as the recruitment age,  $p_k$  is the proportion recruitment to category  $k$  defined to have recruitment, and  $n$  is the number of spatial locations where recruitment occurs.

In the constant recruitment process,  $R_y$ , the number of recruits in year  $y$  is simply the product of the average recruitment  $R_0$  and the annual year class strength multiplier,  $YCS$ , i.e.,

$$R_y = R_0 \times YCS_{y-\text{offset}} \quad (4.7)$$

where  $\text{offset}$  if the number of years offset to link the year class with the year of spawning.

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

$$R_y = R_0 \times YCS_{y-\text{offset}} \times SR(SSB_{y-\text{offset}}) \quad (4.8)$$

where  $\text{offset}$  if 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,  $h$ ,

$$SR(SSB) = \frac{SSB}{B_0} / \left( 1 - \frac{5h-1}{4h} \left( 1 - \frac{SSB}{B_0} \right) \right) \quad (4.9)$$

Note that the Beverton-Holt recruitment process requires a value for  $SSB$  to resolve the stock-recruitment relationship. Here, a derived quantity (see Section 4.9) must be defined that provides the  $SSB$  for the recruitment process.

### 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,

$$\text{element}(i, j, k, l) \leftarrow \text{element}(i, j, k, l-1) \quad (4.10)$$

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

$$\text{element}(i, j, k, \text{age}_{\max}) \leftarrow \text{element}(i, j, k, \text{age}_{\max}) + \text{element}(i, j, k, \text{age}_{\max-1}). \quad (4.11)$$

### 4.7.3 Mortality

Four types of mortality processes are permissible in SPM, constant, annual-rate, event, or biomass-event. These processes remove individuals from the partition, either as a rate (for constant or annual-rate), or as a total number (abundance) or biomass of individuals (for event or biomass-event). 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 annual-rate), then some event mortality in sequence. It is up to the user to specify how this happens.

Mortalities as rates can depend on a layer. Here only one method of dependence is implemented, the multiplicative method. The multiplicative natural mortality method defines that the value of instantaneous mortality applied to the population state within each cell is the product of the layer value, a selectivity-at-age, and the mortality rate.

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,

$$M(a, k, l) = ML_a S_l \quad (4.12)$$

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

$$n'(a, k, l) = n(a, k, l) \exp(-M(a, k, l)) \quad (4.13)$$

Mortality for the annual rate is similar, except that the rate applied in each year is defined as a separate value.

The event mortality types act in a similar manner, except that it removes a specified abundance (number of individuals) or biomass from the partition, rather than applying a mortality 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.

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  $1 \dots K$  for ages  $l = 1 \dots L$  that are subject to event mortality,

$$V(k, l) = S(l)N(k, l) \quad (4.14)$$

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

$$V_{Total} = \sum_K \sum_L V(k, l) \quad (4.15)$$

Hence the exploitation rate to apply is

$$U = \begin{cases} C/V_{total}, & \text{if } C/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \quad (4.16)$$

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

$$R(k, l) = UV(k, l) \quad (4.17)$$

Similarly, the biomass-event mortality type is applied as ... **to be added**.

#### 4.7.4 Category transitions

Category transition processes move individuals between categories. SPM implements two types, the total number and a rate.

The transition type moves a number  $n$  between a source and sink category. The transition process with selectivity  $S$  for source category  $a$  and sink category  $b$  is,

$$\begin{aligned} \text{element}(i, j, a, l) &\leftarrow \text{element}(i, j, a, l) - \frac{nS_l}{\sum_l S_l} \times \text{element}(i, j, a, l) \\ \text{element}(i, j, b, l) &\leftarrow \text{element}(i, j, b, l) + \frac{nS_l}{\sum_l S_l} \times \text{element}(i, j, a, l) \end{aligned} \quad (4.18)$$

The transition rate type moves a proportion  $p$  between a source and sink category. The transition rate process with selectivity  $S$  for source category  $a$  and sink category  $b$  is,

$$\begin{aligned} \text{element}(i, j, a, l) &\leftarrow \text{element}(i, j, a, l) - pS_l \times \text{element}(i, j, a, l) \\ \text{element}(i, j, b, l) &\leftarrow \text{element}(i, j, b, l) + pS_l \times \text{element}(i, j, a, l) \end{aligned} \quad (4.19)$$

#### 4.8 Movement processes

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

$$\text{element}(i, j, k, l) \leftarrow \text{element}(i, j, k, l) + p \times \text{element}(i', j', k, l) \quad (4.20)$$

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  $\text{element}(i', j', k, l)$  as,

$$\text{element}(i', j', k, l) \leftarrow \text{element}(i', j', k, l) - p \times \text{element}(i', j', k, l) \quad (4.21)$$

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., 2008).



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 location to another. A migration can involve one or more categories and movement at age is defined as some proportion multiplies by a selectivity. Migrations are limited in scope to move individuals from one cell to another, and are available to allow compatibility with limited space models such as CASAL Bull et al. (2008).

#### 4.8.2 Adjacent cell movement

The adjacent cell movement simply moves a proportion of individuals to neighbouring cells. It can be applied to a limited range of spatial locations by associating it with a layer.

#### 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 be 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.

#### 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  $\theta_x$  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,

$$P_x = f_1(\theta_1, P_1(x))^{\alpha_1} + f_2(\theta_2, P_2(x))^{\alpha_2} + f_3(\theta_3, P_3(x))^{\alpha_3} + \dots + f_n(\theta_n, P_n(x))^{\alpha_n} \quad (4.22)$$

where  $\alpha_i$  is an arbitrary weighting factor for attribute  $i$ .

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  $a$ ),

$$p(a \rightarrow b) = \frac{P_a}{\sum_{i \in \forall b} P_i} \quad (4.23)$$

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 (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.

### Preference functions

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

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

$$f(x) = x, \text{ where } 0 \leq x \leq 1 \quad (4.24)$$

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

$$f(x|\mu, \sigma) = 2^{-[(x-\mu)/\sigma]^2} \quad (4.25)$$

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

$$f(x|\mu, \sigma_L, \sigma_R) = \begin{cases} 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \quad (4.26)$$

4. The Logistic preference function has dependent variable  $x$  and parameters  $\theta = (a_{50}, a_{t095})$ , and is defined as,

$$f(x|a_{50}, a_{t095}) = 1/[1 + 19^{(a_{50}-x)/a_{t095}}] \quad (4.27)$$

5. The inverse-Logistic preference function has dependent variable  $x$  and parameters  $\theta = (a_{50}, a_{t095})$ , and is defined as,

$$f(x|a_{50}, a_{t095}) = 1 - 1/[1 + 19^{(a_{50}-x)/a_{t095}}] \quad (4.28)$$

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

$$f(x|\lambda) = \exp(-\lambda x), \text{ where } x \geq 0 \text{ and } 0 \text{ otherwise} \quad (4.29)$$

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

$$f(x|N, \lambda) = \begin{cases} 1, & \text{if } 0 \leq x \leq N \\ 1/\left(\frac{x}{N}\right)^\lambda, & \text{if } x \geq N \\ 0, & \text{otherwise} \end{cases} \quad (4.30)$$

#### 4.9 Derived quantities

Derived quantities are values, calculated by SPM as required, that have a single value for each year of the model. Derived quantities can be calculated as either an abundances or as a biomass. Derived quantities are simply the count or sum of cells within some categories, after applying a selectivity, within cells defined by a layer.

Some processes require, as arguments, a population value derived from the population state. These are termed *derived quantities*. For example, a recruitment process may require the amount of spawning stock biomass to resolve the stock-recruit relationship. In this example, the spawning stock biomass could 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. Derived quantities are described further in the population section (Section 4).

#### 4.10 Size-at-age

This is used to determine length frequencies and hence biomass (using a size-weight relationship) of individuals at age/category. There are two alternative growth curves in SPM,

von Bertalanffy where size at age is defined as,

$$\bar{s}(\text{age}) = L_\infty (1 - \exp(-k(\text{age} - t_0))) \quad (4.31)$$

Schnute where size at age is defined as,

$$\bar{s}(\text{age}) = \begin{cases} \left[ y_1^b + (y_2^b - y_1^b) \frac{1 - \exp(-a(\text{age} - \tau_1))}{1 - \exp(-a(\tau_2 - \tau_1))} \right]^{1/b}, & \text{if } a \neq 0 \text{ and } b \neq 0 \\ y_1 \exp \left[ \ln(y_2/y_1) \frac{1 - \exp(-a(\text{age} - \tau_1))}{1 - \exp(-a(\tau_2 - \tau_1))} \right], & \text{if } a \neq 0 \text{ and } b = 0 \\ \left[ y_1^b + (y_2^b - y_1^b) \frac{\text{age} - \tau_1}{\tau_2 - \tau_1} \right]^{1/b}, & \text{if } a = 0 \text{ and } b \neq 0 \\ y_1 \exp \left[ \ln(y_2/y_1) \frac{\text{age} - \tau_1}{\tau_2 - \tau_1} \right], & \text{if } a = 0 \text{ and } b = 0 \end{cases} \quad (4.32)$$

The von Bertalanffy curve is parameterised by  $L_\infty$ ,  $k$ , and  $t_0$ ; the Schnute curve (Schnute, 1981) by  $y_1$  and  $y_2$ , which are the mean sizes at reference ages  $\tau_1$  and  $\tau_2$ , and  $a$  and  $b$  (when  $b = 1$ , this reduces to the von Bertalanffy with  $k = a$ ).

The model can incorporate changes in size-at-age during the year (i.e., growth between fish birthdays) by specifying the growth proportions for each time step of the annual cycle.

#### 4.11 Mean weight

The size-weight parameters  $a$  and  $b$  is calculated as either,

- None: A relationship where the weight of each individual is exactly 1
- Basic: The more usual size-weight relationship, where

$$\text{mean weight} = a(\text{mean size-at-age})^b \quad (4.33)$$

Be careful about the scale of  $a$  — this is easily specified incorrectly. If you provide your catch in tonnes, and your 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 is an option `@report[label].type=weight_at_size` (see Section 7.1.7) that can be used to help check that the units specified are plausible.

If you specify a distribution for the size-at-age relationship, then the mean weight at age is calculated over that distribution, using the following formula, which is exact for lognormal distributions, and a good approximation for a normal distribution (if the c.v. is not large),

$$\text{mean weight} = a \times (\text{mean size at age})^b \times (1 + \text{cv}^2)^{\frac{b(b-1)}{2}} \quad (4.34)$$

where cv is the c.v. of sizes-at-age.

#### 4.12 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 4. SPM implements a number of different parametric forms, including logistic, knife edge, and double normal selectivities. See Section 4.12 for more details.

A selectivity is a function with a different value for each age class (i.e., for each column of the partition). Selectivities are used frequently throughout the SPM population section: for selectivity curves. Selectivities have a number of different parametric forms (types) in SPM and you can use any of these for any selectivity parameter.

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, the you need to specify the categories that it 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% mark at age 5 and 95% mark at age 7. This would be defined by the `Type=Logistic` with parameters  $a_{50} = 5$  and  $a_{95} = (7 - 5) = 2$ . Then the value of the selectivity at age  $x = 3$  is therefore 0.95.

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  $(a_{50} - x)/a_{95} > 5$  then the value of the selectivity at  $x = 0$ , i.e., for  $a_{50} = 5$ ,  $a_{95} = 0.1$ , then the value of the selectivity at  $x = 1$ , without range checking would be  $7.1 \times 10^{-52}$ . With range checking, that value is 0 (as  $(a_{50}x)/a_{95} = 40 > 5$ ).

The available selectivities are;

- Constant
- Knife-edge
- All values
- All values bounded
- Increasing
- Logistic
- Logistic-producing
- Double-normal
- Double-exponential

The available selectivities are described below.

#### 4.12.1 constant

$$f(x) = C \quad (4.35)$$

The constant selectivity has the estimable parameter C.

#### 4.12.2 knife\_edge

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \quad (4.36)$$

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

#### 4.12.3 all\_values

$$f(x) = V_x \quad (4.37)$$

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

#### 4.12.4 all\_values\_bounded

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \quad (4.38)$$

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

#### 4.12.5 increasing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \leq x \leq H \\ f(\alpha), & \text{if } x \geq H \end{cases} \quad (4.39)$$

The increasing ogive has non-estimable parameters  $L$  and  $H$ . The estimable parameters are  $\pi_L, \pi_{L+1} \dots \pi_H$  (but if these are estimated, they should always be constrained to be between 0 and 1).  $\alpha$  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.

#### 4.12.6 logistic

$$f(x) = \alpha / [1 + 19^{(a_{50}-x)/a_{t095}}] \quad (4.40)$$

The logistic selectivity has estimable parameters  $a_{50}$  and  $a_{t095}$ .  $\alpha$  is a scaling parameter, with default value of  $\alpha = 1$ . The logistic selectivity takes values  $0.5\alpha$  at  $x = a_{50}$  and  $0.95\alpha$  at  $x = a_{50} + a_{t095}$ .

#### 4.12.7 logistic\_producing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1)) / (1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \quad (4.41)$$

The logistic-producing selectivity has the non-estimable parameters  $L$  and  $H$ , and has estimable parameters  $a_{50}$  and  $a_{t095}$ . 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  $a_{50}, a_{t095}$ .

#### 4.12.8 double\_normal

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \quad (4.42)$$

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

#### 4.12.9 double\_exponential

$$f(x) = \begin{cases} y_0(y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \leq x_0 \\ y_0(y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \quad (4.43)$$

The double-exponential selectivity has non-estimable parameters  $x_1$  and  $x_2$ , and estimable parameters  $x_0, y_0, y_1$ , and  $y_2$ . It can be ‘U-shaped’. Bounds for  $x_0$  must be such that  $x_1 < x_0 < x_2$ . 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  $(x_0, y_0)$ .

---

## 5 The estimation section

### 5.1 Role of the estimation section

The tasks carried out by the estimation section are:

1. Get the point estimate, i.e., the maximum posterior density estimate (MPD) (see Section 6.3).
2. 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 (Section 6.4).
3. Generate an MCMC sample from the posterior distribution (Section 6.5).
4. Calculate the approximate covariance matrix of the parameters as the inverse of the minimizer's approximation to the Hessian, and the corresponding correlation matrix (Section 6.3).

### 5.2 Specifying the parameters to be estimated

You need to tell SPM which of the estimable parameters are to be estimated by using `@estimate` commands (see Section 9). An `@estimate` command-block looks like,

```
@estimate process[MyRecruitment].r0
lower_bound 1000
upper_bound 100000
prior uniform
```

See Section XXX for instructions on how to generate the parameter name. You have to specify at least one parameter to be estimated if doing an estimation, profile, or MCMC. You still provide values for the parameters to be estimated, and these are used as the starting values for the minimiser (unless you provide alternative starting values 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 XXX). Note that the bounds and prior on an selectivity refer to the selectivity parameters, not the actual values of the selectivity.

You need to estimate all the estimable parameters of an selectivity if you estimate any, but you can fix some of them if you want by setting the lower and upper bound equal. Similarly, if you want to estimate only some elements of a vector, fix the others by setting the bounds equal.

### 5.3 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 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 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 the 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 (probably better) doing profiles of one or more key parameters and seeing if any of the profiled estimates is actually better 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 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 between specified bounds  $l$  and  $u$ , and calculate a point estimate at each value.

By default  $n = 10$ , and  $(l, u) = (\text{lower bound on parameter plus } (range/(2n)), \text{upper bound on parameter less } (range/(2n)))$ . 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 and all the parameter estimates. The initial point estimate is also inserted into the profile (note that this can serve as a 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  $n$ ,  $l$ , and  $u$  values for each. In the case of vector or selectivity parameters, you will also need to specify the element of the vector being profiled.

You can also supply the initial point estimate using `spm -i`, so that SPM doesn't need to do the first minimisation. Be aware that you are supplying the point estimate, not the minimiser starting point to get to the point estimate (as in other situations where `spm -i` is used).

If you have specified multi-phase estimation (see Section XXX), it is only used for the initial point estimate. Subsequent minimisations are done single-phase, as they should start reasonably close to the endpoint and so shouldn't need multiple phases.

If you get an implausible profile, it may be a result of not using enough iterations in the minimiser (in this case, increase `@minimiser.max_iterations` and/or `@minimiser.max_evaluations` and retry), or the convergence criteria may not be strong enough (try setting `@minimiser.gradient_tolerAnce` to a smaller value).

## 5.5 Bayesian estimation

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

Two major steps are best done by an external package, 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 XXX for a better description of the sequence of SPM commands used in a full Bayesian analysis.

SPM uses a straightforward implementation of the Metropolis algorithm (Gelman et al., 1995, Gilks et al., 1994). The Metropolis algorithm attempts to draw a sample from a Bayesian posterior distribution, and calculates the posterior density  $\pi$ , scaled by an unknown constant. The algorithm generates a ‘chain’ or sequence of values. Typically the beginning of the chain is discarded and every  $N$ th 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 is done to prevent the chain from getting ‘stuck’ at the point estimate.)
- Start from a point specified by the user with `spm -i`.

The chain moves in natural space, i.e., no transformations are applied to the estimated parameters. The default proposal distribution is a multivariate normal centred on the current point, with covariance matrix equal to a matrix based on the approximate covariance produced by the minimiser, times some step-size 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_diff` times the difference between the parameters’ lower and upper bound, then the variance is changed, without changing the associated correlations, to  $k = \min\_diff(upper\_bound_i - lower\_bound_i)$ . This is done by setting

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

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_diff` 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_diff` parameter defaults to 0.0001.

- The `@MCMC.step_size` (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), though some experimentation suggested that this may be too high and can lead to a low acceptance rate.

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 (SPM enforces this). The adaptive mechanisms are as follows:

1. You can request that the step size change adaptively at one or more sample numbers. At each adaptation, the step size 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. 1995 for justification.) The new step size is recorded in the objectives file.
2. You can request that the entire covariance matrix change adaptively at one or more sample numbers. At each adaptation, it is replaced with a matrix based on the sample covariance of an earlier section of the chain. The theory here is that the covariance of a portion of chain could potentially be a better estimate of the covariance of the posterior distribution than the inverse Hessian.

The procedure used to choose the sample of points is as follows. First, all points on the chain so far are taken. All points in an initial user-specified period are discarded. The assumption is that the chain will have started moving during this period - if this is incorrect and the chain has still not moved by the end of this period, it is a fatal error and SPM stops. The remaining set of points must contain at least some user-specified number of transitions - if this is incorrect and the chain has not moved this often, it is again a fatal error. If this test is passed, the set of points is systematically sub-sampled down to 1000 points (it must be at least this long to start with).

The variance-covariance matrix of this sub-sample of chain is calculated. As above, correlations greater than `@MCMC.MaxCor` are reduced to `@MCMC.MaxCor`, correlations less than `@MCMC.MaxCor` are increased to `@MCMC.MaxCor`, and very small non-zero variances are increased (`@MCMC.CovarianceAdjustment` and `@MCMC.MinDiff`). The result is the new variance-covariance matrix of the proposal distribution.

The step size parameter is now on a completely different scale, and must also be reset. It is set to a user-specified value (which may or may not be the same as the initial step size). We recommend that some of the step size adaptations are set to occur after this, so that the step size can be readjusted to an appropriate value which gives good acceptance probabilities with the new matrix.

All modified versions of the covariance matrix are printed to the standard output, but only the initial covariance matrix (inverse Hessian) is saved to the objectives file. The number of covariance modifications by each iteration is recorded as a column on the objectives file.

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 written to file.

You have the option to specify that some of the estimated parameters are fixed during 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. If you specify the start of the chain using `spm -i`, these fixed parameters are set to the values in the file.

A multivariate  $t$  distribution is available as an alternative to the multivariate normal proposal distribution. If you request multivariate  $t$  proposals, you may want to change the degrees of freedom from the default of 4. As the degrees of freedom decrease, the  $t$  distribution becomes more heavy tailed. This may lead to better convergence properties.

Given a posterior (sub)sample, `SPM` can calculate a list of output quantities for each sample point (see Section XXX). These quantities can be dumped into a file (using `spm -v`) and read into an external software package where the posterior distributions can be plotted and/or summarised.

The posterior sample can also be used for projections (Section XXX) or simulations (Section XXX), allowing the parameter uncertainty, as expressed in the posterior distribution, can be included into the risk or other output estimates.



---

## 6 The observation section

The objective function is based on the goodness-of-fit of the model to your observations. In the current release of SPM, most observations are different kinds of time series, i.e., data which were recorded for one or more years, in the same format each year. Examples of time series data types include relative abundance indices, commercial catch length frequencies, survey numbers-at-age, etc.,.

Generally, time series must relate to a specified time step and a specified area, and one or more years in which they were recorded.

### 6.1 Types of observations

Five types

Observations of a mortality event proportions of individuals by age class

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

#### 6.1.1 Event mortality proportions-at-age

#### 6.1.2 Proportions-at-age

In the case study, proportions-at-age data from the commercial catch, by SSRU and year (Figure 8), were fitted to the modelled proportions-at-age composition using a multinomial likelihood, i.e.,

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(E_i) \quad (6.1)$$

where  $O_i$  are the observed proportions at age  $i$ ,  $E_i$  are the model expected proportions at age  $i$ , and  $N$  is the effective sample size.

#### 6.1.3 Proportions-by-category

The likelihood for these observations was implemented as a binomial likelihood. Here,

$$\begin{aligned} -\log(L) = & -\sum_i \log(N_i!) - \log((N_i(1 - O_i))!) - \log((N_i O_i)!) + N_i O_i \log(E_i) \\ & + N_i(1 - O_i) \log(1 - E_i) \end{aligned} \quad (6.2)$$

where  $O_i$  are the observed proportions by category  $i$ ,  $E_i$  are the expected proportions by category  $i$ , and  $N$  is the effective sample size.

#### 6.1.4 Abundance

The lognormal likelihood with observations  $O_i$ , c.v.  $c_i$ , and expected values  $qE_i$ , and was,

$$-\log(L) = \sum_{i=1}^n \left( \log(\sigma_i) + 0.5 \left( \frac{\log(O_i/qE_i)}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \quad (6.3)$$

where

$$\sigma_i = \sqrt{\log(1 + c_i^2)} \quad (6.4)$$

For the case study, c.v.s were all assumed to be 0.2.

## 6.2 The objective function

### 6.2.1 The objective function

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

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

where  $\pi$  is the joint prior density of the parameters  $p$ .

Under either estimation method, penalties can be added to the objective function (see Section XXX). You will usually want to use penalties to ensure that the exploitation rate constraints on your fisheries are not breached (otherwise there is nothing to prevent the model from having abundances so low that the recorded catches could not have been taken). A penalty to force the YCS to average to 1 (i.e., to have mean 1) may also be necessary.

### 6.2.2 Likelihoods

### 6.2.3 Priors

### 6.2.4 Penalties

## 6.3 Simulating observations

## 6.4 Pseudo-observations

---

## **7 The report section**

The report section specifies the printouts and other outputs from the model.

### **7.1 Reporting model outputs**

#### **7.1.1 Printing the partition**

#### **7.1.2 Printing derived quantities**

#### **7.1.3 Printing the objective function**

#### **7.1.4 Printing observations, fits, and residuals**

#### **7.1.5 Printing layers**

#### **7.1.6 Printing selectivities**

#### **7.1.7 Verifying the size-weight relationship**





---

## 8 Population command and subcommand syntax

### 8.1 Model structure

**@model1** 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: None

Value: A positive integer,  $n_{rows} > 0$

**ncols** The number of columns  $n_{cols}$  in the spatial structure

Type: Integer

Default: None

Value: A positive integer,  $n_{cols} > 0$

**layer** The label for the base layer

Type: String

Default: None

Value: Must be a label of a `numeric` layer defined by `@layer`

**categories** Labels of the categories (rows) of the population component of the partition

Type: Vector of strings, of length  $1 \dots n_{categories}$

Default: None

Value: Names of categories must be unique

**min\_age** Minimum age of the population

Type: Integer

Default: None

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: None

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

**cell\_length** The length (distance) of one side of a cell

Type: Constant

Default: 1

Value: A positive real number

**size\_at\_age** Define the label of the associated size-at-age relationship for each category

Type: Vector of strings, of length  $1 \dots n_{categories}$

Default: None

Value: Label names must be unique

`initialisation_phases`      Define the labels of the phases of the initialisation

Type: Vector of strings, of length of the number of initialisation phases

Default: None

Value: A valid label defined by `@initialisation_phase`

`initial_year`      Define the first year of the model, immediately following initialisation

Type: Integer

Default: None

Value: Defines the first year of the model,  $\geq 1$ , e.g. 1990

`current_year`      Define the current year of the model

Type: Integer

Default: None

Value: Defines the current year of the model, i.e., the model is run from `@model.first_year` to `@model.current_year`

`final_year`      Define the final year of the model in projections

Type: Integer

Default: None

Value: Defines the final year of the model for use in projections, i.e., the model is run from `@model.first_year` to `@model.current_year`, then projected to `@model.final_year`

`time_steps`      Define the `@time_step` labels (in order that they are applied) to form the annual cycle

Type: String vector

Default: None

Value: Defines the labels of the time steps that are run in each year

## 8.2 Initialisation

The methods for initialisation available are,

- Iterative

Each type of initialisation requires a set of subcommands and arguments specific to that type.

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

`type`      Define the type of initialisation

Type: String

Default: None

Value: A valid type of initialisation

### 8.2.1 @initialisation\_phase[label].type=iterative

years      Define the number of years to run

Type: Integer

Default: None

Value: A non-negative integer

processes      Define the processes (in order of occurrence) to run in each year of the initialisation

Type: String vector

Default: None

Value: A valid process label, from one of @process

## 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: None

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
- Ageing process
- Constant mortality rate process
- Annually varying mortality rate process
- Mortality event (as a number) process
- Mortality event (as a biomass) process
- Category transition process
- Category shift 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: None  
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: None  
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: None  
Value: Valid categories from `@model.categories`

`proportions`     Define the proportion of recruitment that occurs into each category  
Type: Estimable vector of length `categories`  
Default: None  
Value: Proportion of the annual recruitment that is applied to each category

`ages`     Define the ages within each category that receive recruitment  
Type: Integer vector  
Default: None  
Value: The age classes that receive recruitment

`layer`     Name of the layer used to determine where recruitment occurs  
Type: String  
Default: None  
Value: A valid layer as defined by `@layer`. If a numeric layer, then recruitment is in proportion to the layer values. If a logical layer, then recruitment occurs in cells where the layer value is `True`. 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: None  
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: None  
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: None  
Value: Proportion of the annual recruitment that is applied to each category
- ages**      Define the age within each category that receive recruitment  
Type: Integer vector  
Default: None  
Value: The age classes that receive recruitment
- steepness**      Define the Beverton-Holt stock recruitment relationship steepness ( $h$ ) parameter  
Type: Estimable  
Default: None  
Value: Steepness value between 0.2 and 1.0
- sigma\_r**      Define the recruitment variability  $\sigma_R$  in the stock-recruitment relationship for projections  
Type: Estimable  
Default: None
- rho**      Define the autocorrelation  $\rho$  in the recruitment variability in the stock-recruitment relationship for projections  
Type: Estimable  
Default: None
- SSB**      Define the label of the `@derived_quantity` that defines the SSB  
Type: String  
Default: None  
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: None  
Value: Must be a value  $\geq 0$
- YCS\_values**      YCS values  
Type: Estimable vector  
Default: None  
Value: Must be vector  
Note: Special values can be used here, i.e., mean, all
- YCS\_years**      Years for year class strength values

Type: Integer vector  
 Default: None  
 Value: Must be vector of length `YCS_values`  
 Note: Special year ranges (YYYY-YYYY) can be used

`standardise_YCS_year_range`      Years for which the year class strength values are defined to have mean 1.0  
 Type: Integer vector of length 2  
 Default: None  
 Value: Must be vector of length 2, with values of years between `@model.initial` to `@model.current`

`layer`      Name of the layer used to determine where recruitment occurs  
 Type: String  
 Default: None  
 Value: A valid layer as defined by `@layer`. If a numeric layer, then recruitment is in proportion to the layer values. If a logical layer, then recruitment occurs in cells where the layer value is `True`

### 8.4.3 `@process[label].type=ageing`

`categories`      Define the categories that ageing is applied to  
 Type: String vector  
 Default: None  
 Value: Valid categories from `@model.categories`

### 8.4.4 `@process[label].type=constant_mortality_rate`

`m`      Define the constant mortality rate to be applied  
 Type: Estimable  
 Default: None  
 Value: A real number  $\geq 0$  and  $\leq 1$

`categories`      Define the categories that mortality is applied to  
 Type: String vector  
 Default: None  
 Value: Valid categories from `@model.categories`

`selectivities`      Define the selectivities applied to each category  
 Type: String vector  
 Default: None  
 Value: Valid selectivity labels defined by `@selectivity`

`layer`      Name of the layer  
 Type: String  
 Default: None  
 Value: A valid layer as defined by `@layer`. If a numeric layer, then mortality applied is the mortality rate

times the value of the layer. If a logical layer, then mortality occurs in cells where the layer value is `True`. Note that the layer values must be non-negative

#### 8.4.5 `@process[label].type=annual_mortality_rate`

`years` Define the years when the mortality rates are applied  
 Type: Constant vector  
 Default: None  
 Value: Valid model years

`m` Define the mortality rate to be applied for each year  
 Type: Estimable vector  
 Default: None  
 Value: A real number  $\geq 0$  and  $\leq 1$

`categories` Define the categories that mortality is applied to  
 Type: String vector  
 Default: None  
 Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities applied to each category  
 Type: String vector  
 Default: None  
 Value: Valid selectivity labels defined by `@selectivity`

`layer` Name of the layer  
 Type: String  
 Default: None  
 Value: A valid layer as defined by `@layer`. If a numeric layer, then mortality applied is the mortality rate times the value of the layer. If a logical layer, then mortality occurs in cells where the layer value is `True`. Note that the layer values must be non-negative

#### 8.4.6 `@process[label].type=event_mortality`

`categories` Define the categories that the event mortality is applied to  
 Type: String vector  
 Default: None  
 Value: Valid categories from `@model.categories`

`years` Define the years where the mortality even is applied  
 Type: Integer vector  
 Default: None  
 Value: Valid years for the model

`layers` Define the layers that specify the event mortality (as the abundance) in each year

Type: String vector, of length years

Default: None

Value: Valid layers defined by @layer. Note that the layer values must be non-negative

U\_max      Define the maximum exploitation rate

Type: Estimable

Default: None

Value: Must be  $\geq 0$  and  $\leq 1$

selectivities      Define the selectivities applied to each category

Type: String vector

Default: None

Value: Valid selectivity labels defined by @selectivity

penalty      Define the event mortality penalty label

Type: String

Default: None

Value: Valid penalty label defined by @penalty

#### **8.4.7 @process[label].type=biomass.event.mortality**

categories      Define the categories that the event mortality is applied to

Type: String vector

Default: None

Value: Valid categories from @model.categories

size\_at\_age      Define the age-weight relationships for each of the categories that the event mortality is applied to

Type: String vector

Default: None

Value: Valid labels from @size\_at\_age

years      Define the years where the mortality event is applied

Type: Integer vector

Default: None

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

Default: None

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: None

Value: Must be  $\geq 0$  and  $\leq 1$



**selectivities**     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels defined by @selectivity

**penalty**     Define the event mortality penalty label  
Type: String  
Default: None  
Value: Valid penalty label defined by @penalty

#### **8.4.8 @process[label].type=category\_transition**

**from**     Define the categories that are the source of the transition process  
Type: String vector  
Default: None  
Value: A valid list of categories from @model.categories

**selectivities**     Define the selectivities applied to the source categories  
Type: String vector  
Default: None  
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: None  
Value: A valid list of categories from @model.categories

**years**     Define the years where the category transition is applied  
Type: Integer vector  
Default: None  
Value: Valid years for the model

**layers**     Define the layers that specify the event mortality (as N for each cell) in each year  
Type: String vector  
Default: None  
Value: Valid layers defined by @layer. Note that the layer values must be non-negative

**penalty**     Define the penalty to encourage models parameter values away from those which result in not enough individuals to move  
Type: String  
Default: None  
Value: Valid penalty label defined by @penalty

#### 8.4.9 @process[label].type=category\_transition\_rate

**from** Define the category that is the source of the transition process

Type: String

Default: None

Value: A valid category from @model.categories

**selectivities** Define the selectivities applied to the source categories

Type: String vector

Default: None

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: None

Value: A valid category from @model.categories

**proportion** Define the proportion of individuals to move

Type: Estimable

Default: None

Value: A value  $\geq 0$  and  $\leq 1$

**layer** Name of the layer

Type: String

Default: None

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. If a logical layer, then the transition occurs in cells where the layer value is True. Note that the layer values must be non-negative

#### 8.4.10 @process[label].type=migration\_movement

**categories** Define the categories that the migration movement event is applied to

Type: String vector

Default: None

Value: Valid categories from @model.categories

**source\_layer** Define the label of a layer that defines the source cells of the migration movement event

Type: String

Default: None

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: None

Value: A valid layer defined by @layer

proportions      Define the constant multiplier for the proportions that migrate

Type: Estimable

Default: 1.0

Value: A real number between 0 and 1, inclusive

layer      Name of the layer

Type: String

Default: None

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. If a logical layer, then the transition occurs in cells where the layer value is `True`

selectivities      Define the selectivities applied to each category

Type: String vector

Default: None

Value: Valid selectivity labels defined by @selectivity

#### **8.4.11 @process[label].type=adjacent\_cell\_movement**

Not yet implemented. No subcommands have been defined.

#### **8.4.12 @process[label].type=preference\_movement**

categories      Define the categories that the preference function movement is applied to

Type: String vector

Default: None

Value: Valid categories from @model.categories

preference\_functions      Define the labels of the individual preference functions that make up the total preference function

Type: String vector

Default: None

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
- Threshold-biomass

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: None

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: None

Value: A valid layer defined by @layer

**alpha**     Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

### 8.5.2 @preference\_function[label].type=normal

**layer**     Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

**alpha**     Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

**mu**     Defines the  $\mu$  parameter of the normal preference function

Type: Estimable

Default: None

**sigma**     Defines the  $\sigma$  parameter of the normal preference function

Type: Estimable

Default: None

**8.5.3 @preference\_function[label].type=double\_normal**

layer      Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

alpha      Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

mu          Defines the  $\mu$  parameter of the double-normal preference function

Type: Estimable

Default: None

sigma\_l     Defines the  $\sigma_L$  parameter of the double-normal preference function

Type: Estimable

Default: None

sigma\_r     Defines the  $\sigma_R$  parameter of the double-normal preference function

Type: Estimable

Default: None

**8.5.4 @preference\_function[label].type=logistic**

layer      Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

alpha      Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

a50          Defines the  $a_{50}$  parameter of the logistic preference function

Type: Estimable

Default: None

ato95       Defines the  $a_{to95}$  parameter of the logistic preference function

Type: Estimable

Default: None

### 8.5.5 @preference\_function[label].type=inverse\_logistic

layer Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

alpha Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

a50 Defines the  $a_{50}$  parameter of the inverse-logistic preference function

Type: Estimable

Default: None

ato95 Defines the  $a_{to95}$  parameter of the inverse-logistic preference function

Type: Estimable

Default: None

### 8.5.6 @preference\_function[label].type=exponential

layer Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

alpha Defines the multiplicative constant  $\alpha$

Type: Estimable

Default: None

lambda Defines the  $\lambda$  parameter of the exponential preference function

Type: Estimable

Default: None

### 8.5.7 @preference\_function[label].type=threshold

layer Defines the layer which supplies the preference function independent variable

Type: String

Default: None

Value: A valid layer defined by @layer

categories Define the categories are used to calculate the abundance

Type: String vector

Default: None

Value: Valid categories from @model.categories

`selectivities`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels from `@selectivity`

`alpha`     Defines the multiplicative constant  $\alpha$   
Type: Estimable  
Default: None

`n`     Defines the  $N$  parameter of the threshold preference function  
Type: Estimable  
Default: None

`lambda`     Defines the  $\lambda$  parameter of the threshold preference function  
Type: Estimable  
Default: None

#### **8.5.8 `@preference_function[label].type=threshold.biomass`**

`layer`     Defines the layer which supplies the preference function independent variable  
Type: String  
Default: None  
Value: A valid layer defined by `@layer`

`categories`     Define the categories are used to calculate the biomass  
Type: String vector  
Default: None  
Value: Valid categories from `@model.categories`

`selectivities`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels from `@selectivity`

`size_at_age`     Define the age-weight relationships for each of the categories that the biomass is calculated from  
Type: String vector  
Default: None  
Value: Valid labels from `@size_at_age`

`alpha`     Defines the multiplicative constant  $\alpha$   
Type: Estimable  
Default: None

`biomass`     Defines the  $B$  biomass parameter of the threshold biomass preference function  
Type: Estimable  
Default: None

`lambda`     Defines the  $\lambda$  parameter of the threshold biomass preference function  
Type: Estimable  
Default: None

## 8.6 Layers

The available layer types are,

- Numeric
- Categorical
- Distance
- Abundance
- Biomass
- Abundance-density
- Biomass-density
- Meta-layer

**@layer** *label*     Define a layer function with label

`type`     Define the type of layer

Type: String

Default: None

Value: A valid type of layer

### 8.6.1 @layer[label].type=numeric

`data`     Define the values of the layer

Type: Constant vector, with total length  $@model.ncols \times @model.nrows$

Default: None

Value: A vector of values of length equal to the number of elements defined for the spatial structure

### 8.6.2 @layer[label].type=categorical

`data`     Define the values of the layer

Type: Constant vector, with total length  $@model.ncols \times @model.nrows$

Default: None

Value: A vector of values of length equal to the number of elements defined for the spatial structure



### 8.6.3 @layer[label].type=distance

There are no other subcommands for @layer[label].type=distance.

### 8.6.4 @layer[label].type=abundance

**categories**      Define the categories are used to calculate the abundance  
Type: String vector  
Default: None  
Value: Valid categories from @model.categories

**selectivities**      Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels from @selectivity

### 8.6.5 @layer[label].type=biomass

**categories**      Define the categories are used to calculate the biomass  
Type: String vector  
Default: None  
Value: Valid categories from @model.categories

**selectivities**      Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels from @selectivity

**size\_at\_age**      Define the age-weight relationships for each of the categories that the biomass is calculated from  
Type: String vector  
Default: None  
Value: Valid labels from @size\_at\_age

### 8.6.6 @layer[label].type=abundance\_density

**categories**      Define the categories are used to calculate the abundance  
Type: String vector  
Default: None  
Value: Valid categories from @model.categories

**selectivities**      Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels from @selectivity

### 8.6.7 @layer[label].type=biomass\_density

`categories`      Define the categories are used to calculate the biomass

Type: String vector

Default: None

Value: Valid categories from @model.categories

`selectivities`      Define the selectivities applied to each category

Type: String vector

Default: None

Value: Valid selectivity labels from @selectivity

`size_at_age`      Define the age-weight relationships for each of the categories that the biomass is calculated from

Type: String vector

Default: None

Value: Valid labels from @size\_at\_age

### 8.6.8 @layer[label].type=meta

`years`      Define the years

Type: Constant vector, with values for each year of the model

Default: None

`layers`      Define the layer labels for each of the years

Type: String vector, with values for each year specified

Default: None

Condition: Listed layers cannot be @layer[label].type=meta\_layer

`initialisation_layer`      Define the layer label to use during the initialisation

Type: String

Default: None

Condition: Listed layers cannot be @layer[label].type=meta

## 8.7 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: None

Value: A valid type of derived quantity

### 8.7.1 `@derived_quantity[label].type=abundance`

`categories` Define the categories are used to calculate the derived quantity

Type: String vector

Default: None

Value: Valid categories from `@model.categories`

`selectivity` Define the selectivities

Type: String vector

Default: None

Value: Valid selectivity labels from `@selectivity`

`time_step` Define the time step at the end of which, the derived quantity is calculated

Type: String

Default: None

Value: A valid time step label from `@time_step`

`layer` Name of the layer

Type: String

Default: None

Value: A valid layer as defined by `@layer`. If a numeric layer, then value is the sum of the each cell is multiplied by the value of the layer. If a logical layer, then the sum is only for cells where the layer value is True

### 8.7.2 `@derived_quantity[label].type=biomass`

`categories` Define the categories are used to calculate the derived quantity

Type: String vector

Default: None

Value: Valid categories from `@model.categories`

`selectivities` Define the selectivities

Type: String vector

Default: None

Value: Valid selectivity labels from `@selectivity`

`time_step` Define the time step at the end of which, the derived quantity is calculated

Type: String

Default: None

Value: A valid time step label from `@time_step`

`layer`      Name of the layer

Type: String

Default: None

Value: A valid layer as defined by `@layer`. If a numeric layer, then value is the sum of the each cell biomass multiplied by the value of the layer. If a logical layer, then the sum is only for cells where the layer value is `True`

## 8.8 Size-at-age

The individual types of size-at-age relationship available are,

- von Bertalanffy
- Schnute

**`@size_at_age label`**      Define a size-at-age relationship with label

`type`      Define the type of relationship

Type: String

Default: None

Value: A valid type of size-at-age relationship

### 8.8.1 `@size_at_age[label].type=von_Bertalanffy`

`Linf`      Define the  $L_{\infty}$  parameter of the von Bertalanffy relationship

Type: Estimable

Default: None

Value: A positive real number

`k`      Define the  $k$  parameter of the von Bertalanffy relationship

Type: Estimable

Default: None

Value: A positive real number

`t0`      Define the  $t_0$  parameter of the von Bertalanffy relationship

Type: Estimable

Default: None

Value: A real number

`distribution`      Define the distribution of sizes-at-age around the mean

Type: String

Default: Normal

Value: Either normal or lognormal

`by_length`      Specifies if the linear interpolation of c.v.s is a linear function of mean size or of age

Type: Logical

Default: True

Value: If true, the c.v. is a function of length, else a function of age

`cv` Define the c.v. of the distribution of sizes-at-age around the mean

Type: Estimable

Default: None

Value: A positive real number

`growth_proportions` Define the proportion of the year for each time step for evaluating size

Type: Constant vector

Default: None

Value: A vector of values,  $\leq 1$  of length equal to the number of time steps

`size_weight` Define the label of the associated size-weight relationship

Type: String

Default: None

Value: A valid label from `@size_weight`

### 8.8.2 `@size_at_age[label].type=Schnute`

`y1` Define the  $y_1$  parameter of the Schnute relationship

Type: Estimable

Default: None

Value: A positive real number

`y2` Define the  $y_2$  parameter of the Schnute relationship

Type: Estimable

Default: None

Value: A positive real number

`tau1` Define the  $\tau_1$  parameter of the Schnute relationship

Type: Estimable

Default: None

Value: A real number

`tau2` Define the  $\tau_2$  parameter of the Schnute relationship

Type: String

Default: Normal

Value: Either normal or lognormal

`a` Define the  $a$  parameter of the Schnute relationship

Type: String

Default: Normal

Value: Either normal or lognormal

**b** Define the *b* parameter of the Schnute relationship

Type: String

Default: Normal

Value: Either normal or lognormal

**by\_length** Specifies if the linear interpolation of c.v.s is a linear function of mean size or of age

Type: Logical

Default: True

Value: If true, the c.v. is a function of length, else a function of age

**cv** Define the c.v. of the distribution of sizes-at-age around the mean

Type: Estimable

Default: None

Value: A positive real number

**growth\_proportions** Define the proportion of the year for each time step for evaluating size

Type: Constant vector

Default: None

Value: A vector of values,  $\leq 1$  of length equal to the number of time steps

**size\_weight** Define the label of the associated size-weight relationship

Type: String

Default: None

Value: A valid label from @size\_weight

## 8.9 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: None

Value: A valid type of size-weight relationship

### 8.9.1 @size\_weight[label].type=none

There are no other subcommands for @size\_weight[label].type=none.

**8.9.2 @size\_weight[label].type=basic**

- a     Define the  $a$  parameter of the basic relationship  
Type: Estimable  
Default: None  
Value: A positive real number
- b     Define the  $b$  parameter of the basic relationship  
Type: Estimable  
Default: None  
Value: A positive real number

**8.10 Selectivities**

The individual selectivity functions available are,

- Constant
- Knife-edge
- All-values
- All-values-bounded
- Increasing
- Logistic
- Logistic-producing
- Double-normal
- Double-exponential

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: None  
Value: A valid type of selectivity function

**8.10.1 @selectivity[label].type=constant**

- c     Defines the  $C$  parameter of the selectivity function  
Type: Estimable  
Default: None  
Value: A positive real number

### 8.10.2 @selectivity[label].type=knife\_edge

- e Defines the  $E$  parameter of the selectivity function  
Type: Estimable  
Default: None  
Value: A positive real number

### 8.10.3 @selectivity[label].type=all\_values

- v Defines the  $V$  parameters (one for each age class) of the selectivity function  
Type: Estimable vector  
Default: None  
Value: A vector of positive real numbers, of length equal to the number of age classes

### 8.10.4 @selectivity[label].type=all\_values\_bounded

- l Defines the  $L$  parameter of the selectivity function  
Type: Integer  
Default: None  
Value: A positive real number
- h Defines the  $H$  parameter of the selectivity function  
Type: Integer  
Default: None  
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: None  
Value: A vector of positive real numbers, of length equal to the number of age classes from  $L$  to  $H$

### 8.10.5 @selectivity[label].type=increasing

- alpha Defines the  $\alpha$  parameter of the selectivity function  
Type: Estimable  
Default: None  
Value: A positive real number
- l Defines the  $L$  parameter of the selectivity function  
Type: Integer  
Default: None  
Value: A positive real number
- h Defines the  $H$  parameter of the selectivity function



Type: Integer

Default: None

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: None

Value: A vector of positive real numbers, of length equal to the number of age classes from  $L$  to  $H$

### 8.10.6 @selectivity[label].type=logistic

$\alpha$  Defines the  $\alpha$  parameter of the selectivity function

Type: Estimable

Default: None

Value: A positive real number

$a_{50}$  Defines the  $a_{50}$  parameter of the selectivity function

Type: Estimable

Default: None

Value: A positive real number

$a_{t095}$  Defines the  $a_{t095}$  parameter of the selectivity function

Type: Estimable

Default: None

Value: A positive real number

### 8.10.7 @selectivity[label].type=logistic-producing

$\alpha$  Defines the  $\alpha$  parameter of the selectivity function

Type: Estimable

Default: None

Value: A positive real number

$L$  Defines the  $L$  parameter of the selectivity function

Type: Integer

Default: None

Value: A positive real number

$H$  Defines the  $H$  parameter of the selectivity function

Type: Integer

Default: None

Value: A positive real number, must be greater than  $L$

$a_{50}$  Defines the  $a_{50}$  parameter of the selectivity function

Type: Estimable  
Default: None  
Value: A positive real number

ato95     Defines the  $a_{to95}$  parameter of the selectivity function

Type: Estimable  
Default: None  
Value: A positive real number

#### **8.10.8 @selectivity[label].type=double\_normal**

alpha     Defines the  $\alpha$  parameter of the selectivity function

Type: Estimable  
Default: None  
Value: A positive real number

mu        Defines the  $\mu$  parameter of the selectivity function

Type: Estimable  
Default: None

sigma\_l    Defines the  $\sigma_L$  parameter of the selectivity function

Type: Estimable  
Default: None

sigma\_r    Defines the  $\sigma_R$  parameter of the selectivity function

Type: Estimable  
Default: None

#### **8.10.9 @selectivity[label].type=double\_exponential**

alpha     Defines the  $\alpha$  parameter of the selectivity function

Type: Estimable  
Default: None  
Value: A positive real number

x1        Defines the  $x_1$  parameter of the selectivity function

Type: Integer  
Default: None

x2        Defines the  $x_2$  parameter of the selectivity function

Type: Integer  
Default: None

x0        Defines the  $x_0$  parameter of the selectivity function

Type: Estimable

Default: None

$y_0$  Defines the  $y_0$  parameter of the selectivity function

Type: Estimable

Default: None

$y_1$  Defines the  $y_1$  parameter of the selectivity function

Type: Estimable

Default: None

$y_2$  Defines the  $y_2$  parameter of the selectivity function

Type: Estimable

Default: None

## 8.11 Joint selectivities

How this works has yet to be defined...

**@joint.selectivity** *label* Define a joint selectivity

**selectivities** Define the labels of the selectivities to be defined as 'joint'

Type: String vector

Default: None

Value: Valid @selectivity labels



---

## 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: None

Value: A valid label from @minimiser

`MCMC`     The label of the MCMC to use, of doing an MCMC

Type: String

Default: None

Value: A valid label from @MCMC

`profile`     The labels of the profiles to use, if doing a profile

Type: String

Default: None

Value: A valid label from @MPD

`random_seed`     Defines the random number generator seed

*value*     The random number generator seed value

Type: Integer

Default: None

Value: An integer between 0 and 10000 inclusive

### 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`**

`max_iterations`      Define the maximum number of iterations for the minimiser  
Type: Integer  
Default: 1000  
Value: A positive integer

`max_evaluations`      Define the maximum number of evaluations for the minimiser  
Type: Integer  
Default: 4000  
Value: A positive integer

`step_size`      Define the step-size for the minimiser  
Type: Constant  
Default: `1e-6`  
Value: A positive real number

`gradient_tolerance`      Define the gradient tolerance for the minimiser  
Type: Constant  
Default: `0.0002`  
Value: A positive real number

### **9.2.2 `@minimiser[label].type=DE_solver`**

`max_iterations`      Define the maximum number of iterations for the minimiser  
Type: Integer  
Default: 1000  
Value: A positive integer

`max_evaluations`      Define the maximum number of evaluations for the minimiser  
Type: Integer  
Default: 4000  
Value: A positive integer

`step_size`      Define the step-size for the minimiser  
Type: Constant  
Default: `1e-6`  
Value: A positive real number

`gradient_tolerance`      Define the gradient tolerance for the minimiser

Type: Constant

Default: 0.0002

Value: A positive real number

### 9.3 Monte Carlo Markov Chain (MCMC)

Only one method of carrying out MCMCs is available, Monte Carlo Markov Chain 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  $\neq 0$ , 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: None

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  $-\text{max\_correlation}$  are increased to  $-\text{max\_correlation}$

*correlation\_adjustment\_method* Method for adjusting small variances in the covariance

proposal matrix

Type: String

Default: correlation

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

step\_size     Initial step-size (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 step-size 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

## 9.4 Profiles

**@profile** *label*     Define the profile parameters and arguments

parameter     Name of the parameter to be profiled

Type: String

Default: None

Value: Defines the name of the parameter to be profiled

n     Number of values at which to profile the parameter

Type: Integer

Default: 10

Value: Defines the number of values at which to profile the parameter

lower\_bound     lower bound on parameter



Type: Integer

Default: None

Value: Defines the lower bound on the range of the parameter to profile

`upper_bound`      Upper bound on parameter

Type: Integer

Default: None

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: None

Value: A valid SPM parameter name

`same`      Names of the other parameters which are constrained to have the same value

Type: String Vector

Default: None

Value: Defines the names of all the other parameters which are constrained to have the same value as this parameter

`estimation_phase`      Phase at which this parameter should be estimated, in point estimation

Type: Integer

Default: 1

Value: Defines the phase at which this parameter should be freed

`lower_bound`      Lower bounds on this parameter

Type: Constant vector, of length equal to the parameter length

Default: None

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: None

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)

`prior`      Defines the label for the prior for this parameter

Type: String  
Default: No default  
Value: Defines the label of the prior on this parameter

## 9.6 Priors

The available priors are,

- uniform
- uniform-log
- normal
- normal-by-sd
- lognormal
- Beta

**@prior** *label* Define the prior label

*type* Define the type of prior

Type: String  
Default: None  
Value: A valid type of prior

### 9.6.1 @prior[label].type=uniform

The command @prior[label].type=uniform has no other subcommands.

### 9.6.2 @prior[label].type=uniform\_log

The command @prior[label].type=uniform\_log has no other subcommands.

### 9.6.3 @prior[label].type=normal

*mu* Defines the mean  $\mu$  of the normal prior

Type: Constant  
Default: No default  
Value: Defines the mean of the normal prior

*cv* Defines the c.v.  $c$  of the normal prior

Type: Constant  
Default: No default  
Value: Defines the c.v. of the normal prior

#### 9.6.4 `@prior[label].type=normal_by_sd`

`mu` Defines the mean  $\mu$  of the normal by standard deviation prior

Type: Constant

Default: No default

Value: Defines the mean of the normal by standard deviation prior

`sd` Defines the standard deviation  $\sigma$  of the normal by standard deviation prior

Type: Constant

Default: No default

Value: Defines the standard deviation of the normal by standard deviation prior

#### 9.6.5 `@prior[label].type=lognormal`

`mu` Defines the mean  $\mu$  of the lognormal prior

Type: Constant

Default: No default

Value: Defines the mean of the lognormal prior

`cv` Defines the c.v.  $c$  of the lognormal prior

Type: Constant

Default: No default

Value: Defines the c.v. of the lognormal prior

#### 9.6.6 `@prior[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  $\mu$  of the Beta prior

Type: Constant

Default: No default

Value: Defines the mean of the Beta prior

`sd` Defines the standard deviation  $\sigma$  of the Beta prior

Type: Constant

Default: No default

Value: Defines the standard deviation of the Beta prior

## 9.7 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: None

Value: Defines the type of ageing error to use, currently only normal is defined

### 9.7.1 @ageing\_error[label].type=none

The @ageing\_error[label].type=none has no other subcommands.

### 9.7.2 @ageing\_error[label].type=normal

c Parameter of the normal ageing error model

Type: Constant

Default: None

Value: Define the c.v. of misclassification

### 9.7.3 @ageing\_error[label].type=off.by.one

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 misclassifications down and up by 1 year respectively.  $k$  defines the minimum age of fish which can be misclassified - fish under age  $k$  have no ageing error

$p_1$  The  $p_1$  parameter of the off-by-one ageing error model

Type: Constant

Default: None

Value:  $p_1$  and  $p_2$  define the proportions of misclassifications down and up by 1 year respectively.  $k$  defines the minimum age of fish which can be misclassified - fish under age  $k$  have no ageing error

$p_2$  The  $p_2$  parameter of the off-by-one ageing error model

Type: Constant

Default: None

Value: p1 and p2 define the proportions of misclassifications down and up by 1 year respectively. k defines the minimum age of fish which can be misclassified - fish under age k have no ageing error

## 9.8 Defining catchability constants

**@catchability** *label*     Define a catchability constant with *label*

*q*     Value of the *q* parameter

Type: Estimable

Default: None

Value: Defines the value of the catchability *q* parameter, a real positive number

## 9.9 Defining penalties

To be written...



---

## 10 Observation command and subcommand syntax

### 10.1 Observation types

The observation types available are,

Observations of a mortality event proportions of individuals by age class

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

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: None

Value: A valid type of observation

#### 10.1.1 @observation[label].type=event\_mortality\_at\_age

**year**     Define the year that the observation applies to

Type: Integer

Default: None

Value: A positive integer between @model.initial\_year and @model.current\_year

**process\_label**     Define the label of the event mortality process

Type: String

Default: None

Value: A valid label of @process where @process[label].type=event\_mortality

**proportion\_time\_step**     Define the interpolated proportion of the time-step passes that the observation applies to

Type: Constant

Default: 1.0

Value: A real number between 0 and 1, inclusive

**min\_age**     Define the minimum age for the observation

Type: Integer

Default: None

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: None

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: None

Value: Either true or false

layer      Name of the categorical layer used to group the spacial cells for the observation

Type: String

Default: None

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: None

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: None

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: None

Value: A 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: Constant  
Default: 0  
Value: A non-negative real number

`simulate`     Defines if this observation should be simulated when doing simulations  
Type: Switch  
Default: True  
Value: True/False

### 10.1.2 `@observation[label].type=proportions_at_age`

`year`     Define the year that the observation applies to  
Type: Integer  
Default: None  
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: None  
Value: A valid time-step

`proportion_time_step`     Define the interpolated proportion of the time-step passes that the observation applies to  
Type: Constant  
Default: 1.0  
Value: A real number between 0 and 1, inclusive

`categories`     Define the categories  
Type: String vector  
Default: None  
Value: Valid categories from `@model.categories`

`selectivities`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels defined by `@selectivity`

`min_age`     Define the minimum age for the observation  
Type: Integer  
Default: None  
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: None  
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: None  
Value: Either true or false
- `layer`      Name of the categorical layer used to group the spacial cells for the observation  
Type: String  
Default: None  
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: None  
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: None  
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: None  
Value: A 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: Constant  
Default: 0  
Value: A non-negative real number

`simulate`     Defines if this observation should be simulated when doing simulations  
Type: Switch  
Default: True  
Value: True/False

### 10.1.3 `@observation[label].type=proportions.by.category`

`year`     Define the year that the observation applies to  
Type: Integer  
Default: None  
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: None  
Value: A valid time-step

`proportion_time_step`     Define the interpolated proportion of the time-step passes that the observation applies to  
Type: Constant  
Default: 1.0  
Value: A real number between 0 and 1, inclusive

`categories`     Define the categories  
Type: String vector  
Default: None  
Value: Valid categories from `@model.categories`

`categories2`     Define the categories  
Type: String vector  
Default: None  
Value: Valid categories from `@model.categories`

`selectivities`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels defined by `@selectivity`

`selectivities2`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
Value: Valid selectivity labels defined by `@selectivity`

`min_age`     Define the minimum age for the observation

Type: Integer  
Default: None  
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: None  
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: None  
Value: Either true or false

layer      Name of the categorical layer used to group the spacial cells for the observation  
Type: String  
Default: None  
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: None  
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: None  
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: None  
Value: A 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: Constant  
Default: 0  
Value: A non-negative real number

`simulate`     Defines if this observation should be simulated when doing simulations  
Type: Switch  
Default: True  
Value: True/False

#### 10.1.4 `@observation[label].type=abundance`

`year`     Define the year that the observation applies to  
Type: Integer  
Default: None  
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: None  
Value: A valid time-step

`proportion_time_step`     Define the interpolated proportion of the time-step passes that the observation applies to  
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: None  
Value: A valid `@catchability` label

`categories`     Define the categories into which recruitment occurs  
Type: String vector  
Default: None  
Value: Valid categories from `@model.categories`

`selectivities`     Define the selectivities applied to each category  
Type: String vector  
Default: None  
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: None  
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: None

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: None

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: None

Value: A 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: Constant

Default: 0

Value: A non-negative real number

simulate      Defines if this observation should be simulated when doing simulations

Type: Switch

Default: True

Value: True/False

### 10.1.5 @observation[label].type=biomass

year      Define the year that the observation applies to

Type: Integer

Default: None

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: None

Value: A valid time-step

proportion\_time\_step      Define the interpolated proportion of the time-step passes that the

- observation applies to  
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: None  
Value: A valid @catchability label
- categories      Define the categories into which recruitment occurs  
Type: String vector  
Default: None  
Value: Valid categories from @model.categories
- selectivities      Define the selectivities applied to each category  
Type: String vector  
Default: None  
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: None  
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: None  
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: None  
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: None  
Value: A 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: Constant

Default: 0

Value: A non-negative real number

`simulate`      Defines if this observation should be simulated when doing simulations

Type: Switch

Default: True

Value: True/False



---

## 11 Reporting command and subcommand syntax

### 11.1 Reports

The report types available are,

Print the partition

Print a derived quantity

Print the estimated parameters in a vector format (suitable for use with `spm -i`)

Print a summary of the estimated parameters

Print the objective function values

Print an observation values, fits, and residuals

Print the covariance matrix

Print a layer

Print a selectivity's values

Print the weight-at-size using the size-weight relationship

Each type of report requires a set of subcommands and arguments specific to that report.

**@report** *label* Define an output report

*type* Define the type of report

Type: String

Default: None

Value: A valid type of report

#### 11.1.1 @report [*label*] .type=partition

*year* Define the year that the partition report applies to

Type: Integer

Default: None

Value: A positive integer between `@model.initial_year` and `@model.current_year`

*time\_step* Define the time-step that the partition report applies to

Type: Integer

Default: None

Value: A valid time-step

*file\_name* Define the name of the output file where the report is written

Type: String

Default: None

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

### 11.1.2 @report[label].type=initialisation

phase Define the phase of initialisation that the partition report applies to

Type: string

Default: None

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: None

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

### 11.1.3 @report[label].type=derived\_quantity

derived\_quantity Define the label of the derived quantity to print

Type: String

Default: None

Value: A valid label from @derived\_quantity

file\_name Define the name of the output file where the report is written

Type: String

Default: None

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

### 11.1.4 @report[label].type=estimate\_values

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: None  
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

#### **11.1.5 @report[label].type=estimate\_summary**

`file_name` Define the name of the output file where the report is written  
Type: String  
Default: None  
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

#### **11.1.6 @report[label].type=objective\_function**

`file_name` Define the name of the output file where the report is written  
Type: String  
Default: None  
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

#### **11.1.7 @report[label].type=observation**

`observation` Define the label of the observation to print  
Type: String  
Default: None  
Value: A valid label from @Observation

`file_name`     Define the name of the output file where the report is written  
Type: String  
Default: None  
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

### 11.1.8 @report[label].type=layer

`layer`     Define the label of the layer to print  
Type: String  
Default: None  
Value: A valid label from @Layer

`year`     Define the year for the printing of the layer  
Type: Integer  
Default: None  
Value: A positive integer between @model.initial\_year and @model.current\_year

`time_step`     Define the time-step for the printing of the layer  
Type: Integer  
Default: None  
Value: A valid time-step

`file_name`     Define the name of the output file where the report is written  
Type: String  
Default: None  
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

### 11.1.9 @report[label].type=selectivity

`selectivity`     Define the label of the selectivity to print  
Type: String  
Default: None  
Value: A valid label from @Selectivity

`year` Define the year for the printing of the selectivity

Type: Integer

Default: None

Value: A positive integer between `@model.initial_year` and `@model.current_year`

`time_step` Define the time-step for the printing of the selectivity

Type: Integer

Default: None

Value: A valid time-step

`file_name` Define the name of the output file where the report is written

Type: String

Default: None

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

#### **11.1.10 `@report[label].type=random_number_seed`**

`file_name` Define the name of the output file where the report is written

Type: String

Default: None

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

#### **11.1.11 `@report[label].type=weight_at_size`**

`size_weight` Define the label of the size-weight relationship print

Type: String

Default: None

Value: A valid label from `@size_weight`

`sizes` Define the label of the size-weight relationship print

Type: Constant

Default: None

Value: Values of sizes to calculate the size-weight relationship for

`file_name`      Define the name of the output file where the report is written

Type: String

Default: None

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

---

## 12 Other commands and subcommands

**@include** *file*     Include an external file

*file*     The name of the external file to include

Type: string

Default: None

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

### 13.1 An example of a simple $1 \times 1$ non-spatial model

```
categories male female
min_age 2
max_age 25
age_plus_group True
initialisation_phases Phase1
initial_year 1975
current_year 2002
final_year 2007
cell_length 11
time_steps one two three

time_steps 3 # There are three time steps: Oct-Aug, Sep, Sep
recruitment_time 2 # Recruitment occurs in time step 2
recruitment_areas chat # in the area "chat" (the only area defined)
spawning_time 2 # Spawning occurs in time step 2
spawning_part_mort 0.5 # and SSBs are calculated after spawning fish have undergone
    0.5 of the mortality assigned to this time step
spawning_areas chat # Spawning occurs in the area "chat"
spawning_ps 1 # and all mature fish spawn
aging_time 3 # Age incrementation occurs in time step 3
growth_props 1.00 1.00 0.00 # All fish growth occurs in time step 1
M_props 1.00 0.00 0.00 # All natural mortality occurs in time step 1
baranov false # The baranov equation is not used
midmortality_partition weighted_sum
fishery_names chatFishery # The fishery has the label "chatFishery"
fishery_times 1 # and occurs in the first time step
fishery_areas chat # in the area labelled "chat"
n_migrations 0 # No migrations are defined

# Initialisation
@initialisation_phase Phase1
years 200
processes Recruitment Spawning halfM halfM Ageing
```

### 13.2 An example of a simple $10 \times 10$ spatial model

```
categories immature mature spawning
min_age 1
max_age 30
age_plus_group True
initialisation_phases Phase1 Phase2 Phase3
initial_year 1994
Current_year 2008
final_year 2108
cell_length 155.749
time_steps one two

# Initialisation
@initialisation_phase Phase1
years 200
processes Recruitment Spawning Maturation halfM halfM Postspawning Ageing
```

```
@initialisation_phase Phase2
years 1
processes Recruitment Spawning Maturation halfM halfM InitialMoveImmature
      InitialMoveMature InitialMoveSpawning Ageing Postspawning

@initialisation_phase Phase3
years 20
processes Recruitment Spawning Maturation halfM halfM MoveImmature MoveMature
      MoveSpawning Ageing Postspawning

# Annual Cycle
@time_step one
processes Recruitment Spawning Maturation halfM fishing halfM

@time_step two
processes MoveImmature MoveMature MoveSpawning PostSpawning Ageing

# Population processes
@process Ageing
```

### 13.3 An example of a more complex $10 \times 10$ spatial model

```
categories immature mature spawning
min_age 1
max_age 30
age_plus_group True
initialisation_phases Phase1 Phase2 Phase3
initial_year 1994
Current_year 2008
final_year 2108
cell_length 155.749
time_steps one two

# Initialisation
@initialisation_phase Phase1
years 200
processes Recruitment Spawning Maturation halfM halfM Postspawning Ageing

@initialisation_phase Phase2
years 1
processes Recruitment Spawning Maturation halfM halfM InitialMoveImmature
      InitialMoveMature InitialMoveSpawning Ageing Postspawning

@initialisation_phase Phase3
years 20
processes Recruitment Spawning Maturation halfM halfM MoveImmature MoveMature
      MoveSpawning Ageing Postspawning

# Annual Cycle
@time_step one
processes Recruitment Spawning Maturation halfM fishing halfM

@time_step two
processes MoveImmature MoveMature MoveSpawning PostSpawning Ageing

# Population processes
@process Ageing
```

---

## 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 has a number of undocumented helper functions, that could be useful for writing your own analysis functions. See the **R** help for more detail e.g., `help(spm)`



---

## 15 Troubleshooting

### 15.1 Introduction

SPM is a complex system, providing many opportunities for error — either because your parameter files do not correctly specify your model, or because the model you tried to specify does not work as you expect. 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

If you wish to report a bug or problem with SPM, then 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 will not, as a rule, provide help for users of SPM outside of National Institute of Water & Atmospheric Research Ltd. — although we will usually endeavour to try. And, 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 v1.00-2009-03-17 (rev. 3056) Microsoft Windows executable”
2. What operating system or environment are you using? e.g., “IBM-PC Intel CPU running Microsoft Windows XP Service Pack 3”.
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 be 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 Bently (TROPHIA) and Ian Ball (Australian Antarctic Division) for their work that led to the ideas behind the development of this program. Thanks also to Andy McKenzie, Dave Gilbert, and Murray Smith (NIWA) for their helpful discussions with the authors. The SPM logo was designed by Erika Mackay (NIWA).

Much of the structure of SPM, equations, and documentation in this manual draw heavily on similar components of the population model CASAL (Bull et al., 2008). 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 software code, concepts, and documentation from CASAL in SPM.

The development of SPM was funded by the New Zealand Ministry of Fisheries, the Foundation for Research, Science and Technology, and the National Institute of Water & Atmospheric Research Ltd. (NIWA).





---

## 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

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

cell\_length     The length (distance) of one side of a cell

size\_at\_age     Define the label of the associated size-at-age 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

final\_year     Define the final year of the model in projections

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

type     Define the type of initialisation

**@initialisation\_phase[label].type=iterative**

years     Define the number of years to run

processes     Define the processes (in order of occurrence) to run in each year of 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

ages     Define the ages within each category that receive 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  
ages     Define the age within each category that receive recruitment  
steepness     Define the Beverton-Holt stock recruitment relationship steepness ( $h$ ) parameter  
sigma\_r     Define the recruitment variability  $\sigma_R$  in the stock-recruitment relationship for projections  
rho     Define the autocorrelation  $\rho$  in the recruitment variability in the stock-recruitment relationship for projections  
SSB     Define the label of the @derived\_quantity that defines the 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  
YCS\_years     Years for year class strength values  
standardise\_YCS\_year\_range     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=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=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 layer

**@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  
size\_at\_age     Define the age-weight relationships for each of 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=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 event mortality (as N for each cell) in each year  
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  
proportion      Define the proportion of individuals to move  
layer      Name of the layer

**@process[label].type=migration\_movement**

categories      Define the categories that the migration movement event is applied to  
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  
proportions      Define the constant multiplier for the proportions that migrate  
layer      Name of the layer  
selectivities      Define the selectivities applied to each category

**@process[label].type=adjacent\_cell\_movement**

**@process[label].type=preference\_movement**

categories      Define the categories 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  $\alpha$

**@preference\_function[label].type=normal**

layer Defines the layer which supplies the preference function independent variable  
alpha Defines the multiplicative constant  $\alpha$   
mu Defines the  $\mu$  parameter of the normal preference function  
sigma Defines the  $\sigma$  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  $\alpha$   
mu Defines the  $\mu$  parameter of the double-normal preference function  
sigma\_l Defines the  $\sigma_L$  parameter of the double-normal preference function  
sigma\_r Defines the  $\sigma_R$  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  $\alpha$   
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  $\alpha$   
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  $\alpha$   
lambda Defines the  $\lambda$  parameter of the exponential preference function

**@preference\_function[label].type=threshold**

layer Defines the layer which supplies the preference function independent variable  
categories Define the categories are used to calculate the abundance  
selectivities Define the selectivities applied to each category  
alpha Defines the multiplicative constant  $\alpha$   
n Defines the  $N$  parameter of the threshold preference function  
lambda Defines the  $\lambda$  parameter of the threshold preference function

**@preference\_function[label].type=threshold\_biomass**

layer Defines the layer which supplies the preference function independent variable  
categories Define the categories are used to calculate the biomass  
selectivities Define the selectivities applied to each category  
size\_at\_age Define the age-weight relationships for each of the categories that the biomass is

calculated from

alpha      Defines the multiplicative constant  $\alpha$   
biomass     Defines the  $B$  biomass parameter of the threshold biomass preference function  
lambda     Defines the  $\lambda$  parameter of the threshold biomass preference function

**@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

**@layer[label].type=categorical**

data        Define the values of the layer

**@layer[label].type=distance**

**@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

size\_at\_age    Define the age-weight relationships for each of the categories that the biomass is  
calculated from

**@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

size\_at\_age    Define the age-weight relationships for each of the categories that the biomass is  
calculated from

**@layer[label].type=meta**

years        Define the years

layers        Define the layer labels for each of the years

initialisation\_layer    Define the layer label to use during the initialisation

**@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  
selectivity     Define the selectivities  
time\_step     Define the time step at the end of which, the derived quantity is calculated  
layer     Name of the layer

**@derived\_quantity[label].type=biomass**

categories     Define the categories are used to calculate the derived quantity  
selectivities     Define the selectivities  
time\_step     Define the time step at the end of which, the derived quantity is calculated  
layer     Name of the layer

**@size\_at\_age label**     Define a size-at-age relationship with label

type     Define the type of relationship

**@size\_at\_age[label].type=von\_Bertalanffy**

Linf     Define the  $L_{\infty}$  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  
distribution     Define the distribution of sizes-at-age around the mean  
by\_length     Specifies if the linear interpolation of c.v.s is a linear function of mean size or of age  
cv     Define the c.v. of the distribution of sizes-at-age around the mean  
growth\_proportions     Define the proportion of the year for each time step for evaluating size  
size\_weight     Define the label of the associated size-weight relationship

**@size\_at\_age[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  $\tau_1$  parameter of the Schnute relationship  
tau2     Define the  $\tau_2$  parameter of the Schnute relationship  
a     Define the  $a$  parameter of the Schnute relationship  
b     Define the  $b$  parameter of the Schnute relationship  
by\_length     Specifies if the linear interpolation of c.v.s is a linear function of mean size or of age  
cv     Define the c.v. of the distribution of sizes-at-age around the mean  
growth\_proportions     Define the proportion of the year for each time step for evaluating size  
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 relationship

b Define the  $b$  parameter of the basic 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  $\alpha$  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  $\alpha$  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  $\alpha$  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  $\alpha$  parameter of the selectivity function

mu Defines the  $\mu$  parameter of the selectivity function

sigma\_l Defines the  $\sigma_L$  parameter of the selectivity function

sigma\_r Defines the  $\sigma_R$  parameter of the selectivity function

**@selectivity[label].type=double\_exponential**

alpha     Defines the  $\alpha$  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

**@joint\_selectivity** *label*     Define a joint selectivity

selectivities     Define the labels of the selectivities to be defined as 'joint'

## 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, of doing an MCMC  
profile     The labels of the profiles to use, if doing a profile  
random\_seed     Defines the random number generator seed

**@Minimiser** *label*     Define the an minimiser estimator with label

type     Define the type of minimiser

**@minimiser[label].type=numerical\_differences**

max\_iterations     Define the maximum number of iterations for the minimiser  
max\_evaluations     Define the maximum number of evaluations for the minimiser  
step\_size     Define the step-size for the minimiser  
gradient\_tolerance     Define the gradient tolerance for the minimiser

**@minimiser[label].type=DE\_solver**

max\_iterations     Define the maximum number of iterations for the minimiser  
max\_evaluations     Define the maximum number of evaluations for the minimiser  
step\_size     Define the step-size for the minimiser  
gradient\_tolerance     Define the gradient tolerance for the minimiser

**@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**

`correlation_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

`step_size`      Initial step-size (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

**@profile** *label*      Define the profile parameters and arguments

`parameter`      Name of the parameter to be profiled

`n`      Number of 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?

`prior`      Defines the label for the prior for this parameter

**@prior** *label*      Define the prior label

`type`      Define the type of prior

**@prior[label].type=uniform**

**@prior[label].type=uniform\_log**

**@prior[label].type=normal**

`mu`      Defines the mean  $\mu$  of the normal prior

`cv`      Defines the c.v. *c* of the normal prior

**@prior[label].type=normal\_by\_sd**

`mu`      Defines the mean  $\mu$  of the normal by standard deviation prior

`sd`      Defines the standard deviation  $\sigma$  of the normal by standard deviation prior

**@prior[label].type=lognormal**

`mu`      Defines the mean  $\mu$  of the lognormal prior

`cv`      Defines the c.v. *c* of the lognormal prior

**@prior[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  $\mu$  of the Beta prior  
sd Defines the standard deviation  $\sigma$  of the Beta prior

**@ageing\_error** *label* Define ageing error with *label*

*type* The type of ageing error

**@ageing\_error[*label*].type=none**

**@ageing\_error[*label*].type=normal**

*c* Parameter of the normal ageing error model

**@ageing\_error[*label*].type=off\_by\_one**

*k* The  $k$  parameter of the off-by-one ageing error model

*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

**@catchability** *label* Define a catchability constant with *label*

*q* Value of the  $q$  parameter

### 17.3 Observation command and subcommand syntax

**@observation** *label* Define an observation

*type* Define the type of observation

**@observation[*label*].type=event\_mortality\_at\_age**

*year* Define the year that the observation applies to

*process\_label* Define the label of the event mortality process

*proportion\_time\_step* Define the interpolated proportion of the time-step passes that the observation applies to

*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

*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]*

*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

*simulate* Defines if this observation should be simulated when doing simulations

**@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\_time\_step** Define the interpolated proportion of the time-step passes that the observation applies to  
**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  
**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]  
**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  
**simulate** Defines if this observation should be simulated when doing simulations

**@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\_time\_step** Define the interpolated proportion of the time-step passes that the observation applies to  
**categories** Define the categories  
**categories2** Define the categories  
**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  
**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  
**simulate** Defines if this observation should be simulated when doing simulations

**@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\_time\_step** Define the interpolated proportion of the time-step passes that the

observation applies to

`catchability` Define the catchability constant label for the observation  
`categories` Define the categories into which recruitment occurs  
`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  
`simulate` Defines if this observation should be simulated when doing simulations

**@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_time_step` Define the interpolated proportion of the time-step passes that the observation applies to  
`catchability` Define the catchability constant label for the observation  
`categories` Define the categories into which recruitment occurs  
`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  
`simulate` Defines if this observation should be simulated when doing simulations

## 17.4 Reporting command and subcommand syntax

**@report label** Define an output report

`type` Define the type of report

**@report [label].type=partition**

`year` Define the year 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=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=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=estimate\_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=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=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=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=layer**

`layer` Define the label of the layer to print  
`year` Define the year 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

`year` Define the year for the printing of the selectivity  
`time_step` Define the time-step for the printing of the selectivity  
`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=weight\_at\_size**

`size_weight` Define the label of the size-weight relationship print  
`sizes` Define the label of the size-weight relationship 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

## 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, and R. Bian. CASAL C++ Algorithmic Stock Assessment Laboratory): CASAL user manual v2.20-2008/02/14. Technical report, NIWA, 2008.
- J. E. Dennis Jr and R.B. Schnabel. *Numerical methods for unconstrained optimisation and nonlinear equations*. Classics in Applied Mathematics. Prentice Hall, 1996.
- 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.
- 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.
- 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

Common Public License Version 1.0

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

- Abundance layers, 16
- Abundance-density layers, 17
- Adjacent cell movement, 19, 23
- Ageing, 19, 20
- Annual cycle, 3, 18, 19
- Base layer, 16
- Basic size-weight relationship, 26
- Beverton-Holt recruitment, 20
- Biomass layers, 17
- Biomass-density layers, 17
- CASAL, 3
- Categorical layers, 16
- Category transition, 19
- Category transitions, 22
- Citation, 1
- Citing SPM, 1
- Classification layers, 16
- Command
  - Include files, 11
- Command block format, 10
- Command line arguments, 7, 8
- Commands, 9
- Commands
  - Subcommands, 10
- Comments, 11
- Common Public License, 1
- Constant preference function, 24
- Covariate layers, 16
- Derived quantities, 3, 25
- Distance layers, 16
- Double-normal preference function, 24
- Estimable parameters, 7
- Estimated parameters, 4, 7
- Estimation section, 4
- Event mortality, 21
- Exponential preference function, 25
- Including external files, 7
- Initialisation, 17
- Initialisation
  - phases, 18
- Input configuration file, 4, 7
- Input configuration file syntax, 9
- Inverse-Logistic preference function, 24
- Layers, 15
- Layers
  - the base layer, 16
- Linux, 1, 7
- Logistic preference function, 24
- Maximum exploitation rate, 22
- Mean weight, 26
- Meta-layers, 17
- Microsoft Windows, 1, 7
- Migration, 19
- Migration movement, 23
- Model initialisation, 18
- Model overview, 3
- Model years, 18
- Mortality, 19, 21
- Movement, 19, 22
- Movement processes, 19, 22
- Normal preference function, 24
- Numeric layers, 16
- Observation section, 4
- Optional command line arguments, 9
- Output header information, 8
- Partition, 3
- Population processes, 19, 20
- Population section, 4, 13
- Population structure, 14
- Preference function, 23
- Preference functions, 24
- Preference functions
  - Constant, 24
  - Double-Normal, 24
  - Exponential, 25
  - Inverse-Logistic, 24
  - Logistic, 24
  - Normal, 24
  - Threshold, 25
- Preference movement, 19, 23
- Processes, 3, 19
- Projection years, 17
- Projections, 19
- Recruitment, 19, 20
- Recruitment
  - Beverton-Holt, 20
  - Constant, 20

Redirecting standard error, 8  
Redirecting standard out, 8  
Report section, 5  
Run years, 17  
Running SPM, 7

Schnute growth curve, 25  
Selectivities, 26  
Size-at-age, 25  
Software license, 1  
Spatial structure, 13  
Standard error, 7  
Standard out, 7  
State, 3  
Structure, 3  
Subcommand argument type, 10  
System requirements, 1

Tasks, 7  
The estimation section, 4  
The observation section, 4  
The population section, 4  
The report section, 5  
The total preference function, 23  
Threshold preference function, 25  
Time sequences, 17  
Time steps, 3

Using SPM, 7

Version, 1  
von Bertalanffy growth curve, 25