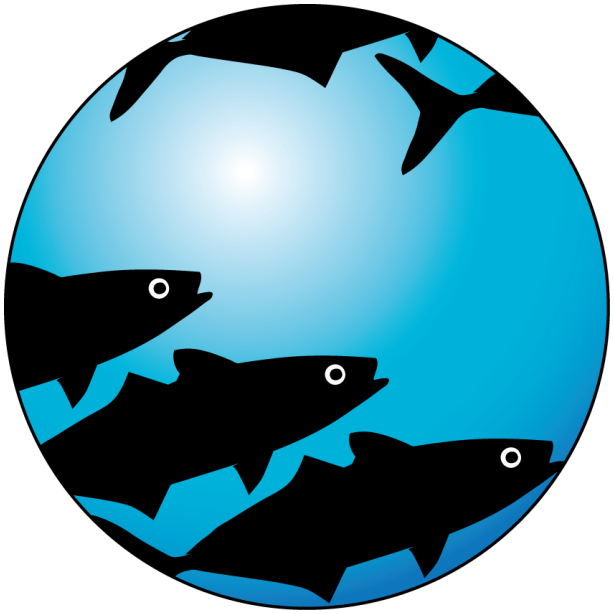
Spatial Population Model User Manual
SPM v1.00-2009-05-21 (rev. 3374)

Alistair Dunn, Scott Rasmussen

May 2009



Spatial Population Model User Manual (modified: 2009-05-21)
for use with SPM v1.00-2009-05-21 (rev. 3374)

Citation:
Dunn, A.; Rasmussen, S. (2009) Spatial Population Model User Manual, SPM v1.00-2009-05-21 (rev. 3374) . National Institute of Water & Atmospheric Research Ltd. *Unpublished report*. 157 p.

Contents

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 2009-05-21) describes SPM v1.00-2009-05-21 (rev. 3374) . 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 v1.00-2009-05-21 (rev. 3374) . National Institute of Water & Atmospheric Research Ltd. *Unpublished report*. 157 p.

1.3 Software license

1.4 System requirements

SPM is available for most IBM compatible machines running 32-bit or 64-bit Linux and most recent 32-bit Microsoft Windows operating systems.

Several of SPMs tasks are highly computer intensive and a fast 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 was compiled on Linux using gcc, the C/C++ compiler developed by the GNU Project. The 32-bit Linux version was compiled using gcc version 4.1.2 20070115 (prerelease) (SUSE Linux), the 64-bit Linux version used 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 was compiled using Mingw32 gcc 3.4.5, and should run on most 32-bit WindowsXP or newer systems. There are no plans to port SPM to Microsoft Windows 64-bit platforms. The Microsoft Windows installer was built using the Nullsoft Scriptable Install System.

SPM uses two minimisers — the first is closely based on the main algorithm of Dennis Jr and Schnabel (1996), and which which uses finite difference gradients, and the second is an implementation of the differential evolution solver (Storn and Price, 1995), and based on code by Lester E. Godwin of PushCorp, Inc. The random number generator used by SPM uses an implementation of the Mersenne twister random number generator (Matsumoto and Nishimura, 1998). This, the command line functionality, matrix operations, and a number of other functions use the BOOST C++ library (Version 1.38.0).

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 either as a part of the installation, or on request from the authors.

2 Model overview

2.1 Introduction

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

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 total number of individuals in a category labelled mature at some point in the annual cycle may be a derived quantity. The state is the combination of the partition and the derived quantities at some instance in time. Changes to the state occur by the application of *processes*. Additions to the vectors of derived quantities occur when a model is requested to add a value to each derived quantity vector.

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

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

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

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

An SPM model can be parametrised by both population processes (for example, ageing, recruitment, and mortality) and movement processes. Movement is parameterised by either adjacent cell movements (*not yet implemented*), between cell migrations(*not yet implemented*), or by global movements as a function of known attributes at each spatial location (termed preference functions — see later). SPM is designed to be flexible and to allow for the estimation of both population and movement parameters from local or aggregated spatially explicit observations.

The population structure of SPM follows the usual population modelling conventions and is similar to those implemented in other population models, for example CASAL (Bull et al., 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)

A model is implemented in SPM using an input configuration file, which is a complete description of the model structure (i.e., spatial and population processes), observations, estimation methods, and reports (outputs) requested. SPM runs from a console window on Microsoft Windows or from a text terminal on Linux. A model can be either *run*, estimable parameters can be *estimated* or *profiled*, *MCMC* distributions calculated, and these estimates can be *projected* (*not yet implemented*) into the future or used by SPM as parameters of 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 structure, mathematical equations used, and command and subcommand syntax and 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). The input configuration file completely describes a model implemented in SPM. See Sections 8, 9, 10, and 11 for details and specification of SPMs command and subcommand syntax within the input configuration file.

2.2 The population section

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

2.3 The estimation section

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

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 model parameters are to be estimated, priors, starting values, and minimiser control values.

Penalties and priors act as constraints on the estimation. They can either encourage or discourage (depending on the specific implementation) parameter estimates that are 'near' some value, and hence influence the estimation process. For example, a penalty can be included in the objective function to discourage parameter estimates that lead to models where the recorded 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 (Section 6). Observations are data which allow us to make inferences about unknown parameters. The observation section specifies the observations, their errors, likelihoods, and when the observations occur. Examples include relative or absolute abundance indices, proportions-at-age frequencies, etc,. Estimation uses the observations to find values for each of the estimated parameters so that each observation is 'close' (in some mathematical sense) to a corresponding expected value.

2.5 The report section

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

3 Running SPM

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

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

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

3.2 The input configuration file

The input configuration file is made up of four broad sections; the description of the population structure and parameters (the population section), the estimation methods and variables (the estimation section), the observations and their associated likelihoods (the observation section), and the outputs and reports that SPM will return (the report section). The input configuration file is 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 is you're using SPM inside a batch job software, i.e.

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

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

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

And, on some Microsoft Windows systems (e.g., 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`. (*not yet implemented*).

-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`. (*not yet implemented*).

-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 11.1.7 for details about the format of `file`.

-t *number* Number of *threads* to run (i.e., number of processors available for use). (*not yet implemented*).

-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.

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 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. (*not yet implemented*).

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

`process[MyRecruitment].r0`

3.6 SPM exit status values

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

4 The population section

4.1 Introduction

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

The population section consists of several components, including;

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

4.2 Spatial structure

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

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

The dimensions of the spatial grid are user defined but must be at least a 1×1 grid (i.e., a single spatial cell). But the largest spatial structure allowed by SPM is a grid of 1000×1000 cells. 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.

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

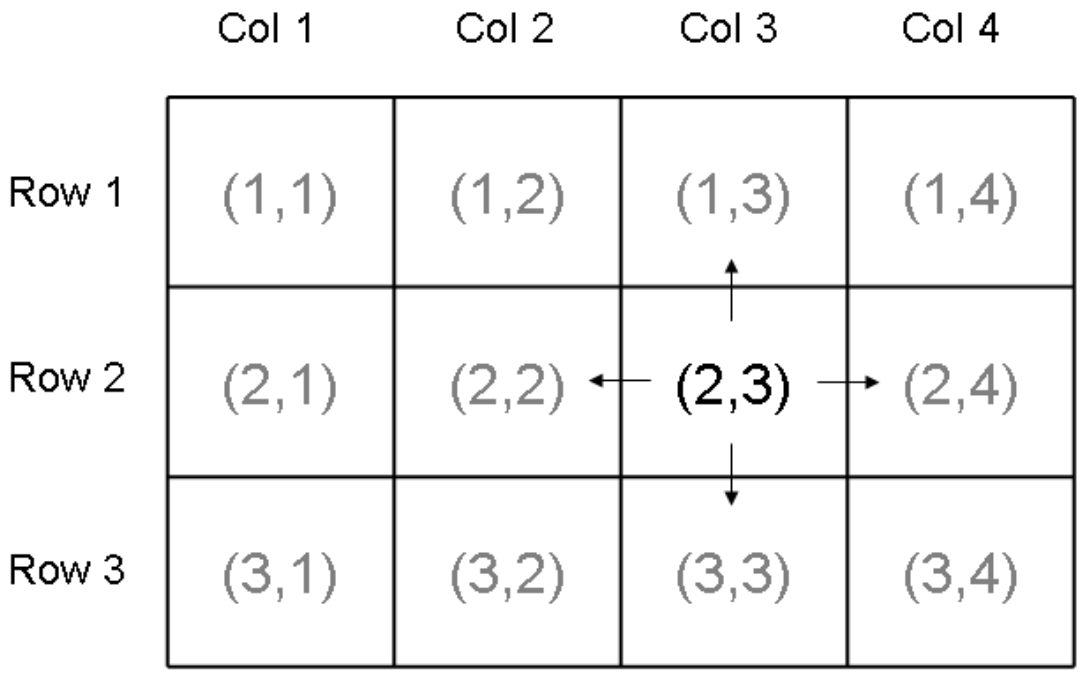


Figure 4.1: An illustration of the *square* spatial structure

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

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

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 *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 ≥ 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,

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

where x and y represent the x- and y-coordinates of a and b respectively, and λ is an arbitrary scaler representing the length of one side of the square. 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.

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

`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 (*not yet implemented*) is the sum of the biomass of individuals within cell a in categories k, with selectivity S_l at age l, and mean weight w_{kl}

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

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

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

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

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

7. Biomass-density layers: The biomass-density layer (*not yet implemented*) is the density of the biomass of individuals within cell a with area A_a in categories k, with selectivity S_l at age l, and mean weight w_{kl},

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

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

8. Meta-layers: In addition to the above types of layer, `SPM` defines a special type of layer known as a *meta-layer* (*not yet implemented*)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 Initialisation

Model initialisation can occur in several phases, each which iterates through a number of years carrying out the population and/or spatial processes defined for that phase. At the end of the initialisation step, SPM runs through the model years carrying out processes in the order defined in the annual cycle, and can evaluate expected values of observations in order to calculate likelihoods, project forward to determine future states (*not yet implemented*), or simulate observations from the current state.

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 take many iterations, and 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.

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

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

Hence, for initialisation you need to define;

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

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 (*not yet implemented*) 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

`SPM`can project, from a set of parameter estimates, the state of the model into the future (*not yet implemented*). In a projection run, the model is initialised and run through the model years from `initial` to the `current`. Then, the partition is run from `current` to `final`.

4.6 Processes

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

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

The population processes include recruitment, ageing, mortality events (e.g., natural and exploitation) and category transition processes (i.e., processes that move individuals between

categories, while preserving their age structure). See Section 4 for a complete list of available processes.

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

SPM implements three different types of movement processes;

1. A migration movement rate of cohorts between any two locations, and is roughly analogous to movements between areas as implemented in other population models, such as CASAL (Bull et al., 2008). (*not yet implemented*)
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). (*not yet implemented*)
3. Movement parametrised as a probability density function. Here, the key underlying idea is that the spatial distribution of cohorts at any point in time and at any location can be represented as a density function based on attributes of that location, local abundance, and/or distance from their previous location (Bentley et al., 2004a,b).

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

4.7 Population processes

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

4.7.1 Recruitment

Recruitment processes are defined as 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

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

where age is the age defined as the recruitment age, p_k is the proportion recruitment to category k defined to have recruitment, n is the number of spatial locations where recruitment occurs, and the recruitment to each cell is scaled to be proportional to the value of the layer in that cell.

In the constant recruitment process, R_y, the number of recruits in year y is simply the average recruitment R_0, i.e.,

$$R_y = R_0 \tag{4.7}$$

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

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

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.,

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

where $offset$ 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,

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

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,

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

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

$$\text{element}(i,j,k,age_{max}) \leftarrow \text{element}(i,j,k,age_{max}) + \text{element}(i,j,k,age_{max-1}). \tag{4.11}$$

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

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

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

4.7.3 Mortality

Four types of mortality processes are permissible in SPM, constant, annual-rate (*not yet implemented*), event, or biomass-event (*not yet implemented*). 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 (yet) 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 morality rate M, and a selectivity-at-age S_l at age l for some user-defined categories k then,

$$M(a,k,l) = ML_aS_l \tag{4.12}$$

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

$$n'(a,k,l) = n(a,k,l)\exp(-M(a,k,l)) \tag{4.13}$$

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

For example, to specify an constant annual mortality rate ($M = 0.2$) for categories 'immature' and 'mature', then,

```
@process NaturalMortality
type constant_mortality_rate
categories immature mature
M 0.2 0.2
selectivities One One
```

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

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

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. Event mortality processes require a penalty function to discourage parameter values that do not allow a the defined number of individuals to be removed. Here, the model penalises those parameter estimates that result in an insufficient number of individuals in defined categories (after applying selectivities). See Section 5.8 for more information on specifying penalties.

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

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

$$V(k,l) = S(l)N(k,l) \tag{4.14}$$

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

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

Hence the exploitation rate to apply is

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

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

$$R(k,l) = UV(k,l) \tag{4.17}$$

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

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

4.7.4 Category transitions

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

The category transition process moves a number n between some source and sink category (or categories). This process may be used, for example, to implement a 'tagging' process for mark-

recapture data. We define the transition process with selectivity S for source category a and sink category b as,

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

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

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

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

Note that this syntax can be used to combine individuals, simply by repeating a category label when specifying the sink categories.

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,

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

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

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

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

4.8 Movement processes

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

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

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 element(i',j',k,l) as,

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

`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 (*not yet implemented*) 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 (*not yet implemented*) 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 to reduced, and results in a movement process that is some function of some underlying property of each location. For example, if we assume that movement between areas was a function of the Euclidean distance between areas, we could model movement between any two areas as a linear decay or exponential decay function (Bentley et al., 2004a). Alternately, if distribution and density were correlated with bathymetric depth for a marine organism, we might model the movement and distribution as a function of depth.

The total preference function

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

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

where α_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),

$$p(a \rightarrow b) = \frac{P_a}{\sum\limits_{i \in \forall b} P_i} \tag{4.23}$$

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,

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

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

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

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

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

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

$$f(x|a_{50},a_{to95}) = 1/[1+19^{(a_{50}-x)/a_{to95}}] \tag{4.27}$$

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

$$f(x|a_{50},a_{to95}) = 1 - 1/[1+19^{(a_{50}-x)/a_{to95}}] \tag{4.28}$$

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

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

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

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

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 abundance 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).

For example, to define a derived quantity (say spawning stock biomass, SSB) for a model, evaluated at the end of the first time step over areas defined by a layer as the spawning ground but counting all 'mature' individuals, we might use the syntax,

```
@derived_quantity SSB
categories mature
selectivity One
time_step 1
layer spawning_ground
```

4.10 Size-at-age relationship

Size-at-age relationships (*not yet implemented*) are used to determine length frequencies, and hence derive a weight-at-age (in conjunction with the 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,

$$\bar{s}(age) = L_\infty \left(1 - \exp\left(-k\left(age - t_0\right)\right)\right) \tag{4.31}$$

Schnute where size at age is defined as,

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

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

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 Size-weight relationship

The size-weight relationship (*not yet implemented*), with 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

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

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 7.17) 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),

$$\text{mean weight} = a \times (\text{mean size at age})^b \times \left(1 + \text{c}^2\right)^{\frac{b(b-1)}{2}} \tag{4.34}$$

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

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

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

The available selectivities are;

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

The available selectivities are described below.

4.12.1 constant

$$f(x) = C \tag{4.35}$$

The constant selectivity has the estimable parameter C.

4.12.2 knife_edge

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

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

4.12.3 all_values

$$f(x) = V_x \tag{4.37}$$

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

4.12.4 all_values_bounded

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

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

4.12.5 increasing

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

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

4.12.6 logistic

$$f(x) = \alpha / [1 + 19^{(a_{50} - x)/a_{to95}}] \tag{4.40}$$

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

4.12.7 `logistic_producing`

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

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

4.12.8 `double_normal`

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

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

4.12.9 `double_exponential`

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

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

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

The estimation section defines The objective function is based on a goodness-of-fit measure of the model to observations, priors and penalties. The observation section describes the objective function, observations, priors and penalties.

5.2 The objective function

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

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

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

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

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), penalties on category transitions (to ensure there are enough individuals to move), and possibly penalties to encourage estimated values to be similar, smoothed, etc.

5.3 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 3.5.5 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 run. You still provide values for the parameters to be estimated, and these are used as the starting values for the minimiser. However, these may be overwritten if you provide a set of alternative starting values (i.e., using `spm -i`, see Section 3.4).

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

Phased estimation of parameters is not yet implemented.

5.4 Point estimation

Point estimation is invoked with `spm -e`. Mathematically, it is an attempt to find a minimum of the objective function. SPM has two algorithms for solving (minimising) the optimisation problem. The first uses a quasi-Newton minimiser built which is a slightly modified implementation of the main algorithm of Dennis Jr. & Schnabel (Dennis Jr and Schnabel, 1996), while the second uses a genetic algorithm developed by Storn & Price (Storn and Price, 1995), the differential evolution minimiser .

5.4.1 The numerical differences minimiser

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

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

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

We want to stress that this is a local optimisation algorithm trying to solve a global optimisation problem. What this means is that, even if you get a ’successful convergence’ message, your solution may be only a local minimum, not a global one. To diagnose this problem, try doing multiple runs from different starting points and comparing the results, or doing profiles of one or more key parameters and seeing if any of the profiled estimates finds a better optimum than than the original point estimate.

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

calculated. Be aware that

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

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

5.4.2 The differential evolution minimiser

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

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

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

5.5 Posterior profiles

If profiles are requested `spm -p`, `SPM` will first calculate a point estimate. For each scalar parameter or, in the case of vectors or selectivities, the element of the parameter to be profiled, `SPM` will fix its value at a sequence of n evenly spaced numbers between specified bounds l and u, and calculate a point estimate at each value.

By default $n = 10$, and $(l,u) =$ (lower bound on parameter plus $(range/(2n))$, upper bound on parameter less $(range/(2n))$. Each minimisation starts at the final parameter values from the previous resulting value of the parameter being profiled. `SPM` will report the objective function for each parameter value. Note that an initial point estimate should be compared with the profile,

not least to check that none of the other points along the profile have a better objective function value than the initial 'minimum'.

You specify which parameters are to be profiled, and optionally n, l, and u values for each. In the case of vector parameters, you will also need to specify the element of the vector being profiled.

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

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

5.6 Bayesian estimation

SPM can use a Monte Carlo Markov Chain (*not yet implemented*) 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 nth set of values).

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

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

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

SPM uses a straightforward implementation of the Metropolis 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 π, scaled by an unknown constant. The algorithm generates a 'chain' or sequence of values. Typically the beginning of the chain is discarded and every Nth element of the remainder is taken as the posterior sample. The chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

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

An initial point estimate is produced before the chain starts, which is done so as to calculate the approximate covariance matrix of the estimated parameters (as the inverse Hessian), and may also be used as the starting point of the chain.

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

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

- Start from the point estimate.
- Start from a random point near the point estimate (the point is generated from a multivariate normal distribution, centred on the point estimate, with covariance equal to the inverse Hessian times a user-specified constant). This 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 $k = \text{min\_diff}(upper\_bound_i - lower\_bound_i)$. This is done by setting

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

    for $i \neq j$, and $\text{var}(i)' = k$
  - if `@MCMC.adjustment_method=correlation`: that if the variance of the *i*th parameter is non-zero and less than `@MCMC.min_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. (Gelman et al., 1995) for justification.)
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 recorded.

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

The posterior sample can also be used for projections (Section 4.5.4) or simulations (Section sec:simulation-observations) with the values supplied using `spm -i` *`file`*.

5.7 Priors

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

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

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

1. Uniform

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

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

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

3. Normal with mean μ and c.v. c

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

4. Normal with mean μ and standard deviation σ

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

5. Lognormal with mean μ and c.v. c

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

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

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

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

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

5.8 Penalties

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

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

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

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

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

6 The observation section

6.1 Observations and likelihoods

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

SPM allows the following types of observations;

Observations of proportions by age class within categories

Observations of proportions between categories within age classes

Relative and absolute abundance/biomass observations

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

6.2 Proportions-at-age observations

Proportions-at-age observations are observations of either the relative number of individuals at age or relative biomass at age.

The observation is supplied for a given year and time step, for some selected age classes of the population (i.e., for a range of ages multiplied by a selectivity), for categories aggregated over a set of spatial cells. Note that the observations at age can include a plus group, which must be less than or equal to the maximum age defined for the partition.

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

Proportions-at-age observations can be supplied for a single category, aggregated across categories, or be proportions of multiple categories. For example, for a model with the two categories *male* and *female*, we might supply either (i) observations of the proportions of males (or alternately female) within each age class; (ii) proportions of total individuals (males + females) at each age class, or (iii) the proportions of individuals for both male and female categories simultaneously. In addition, each category must have an associated selectivity, defined by `selectivities`.

The way the categories of the observation are defined specifies which of these alternatives to use. For example, to specify that the observations are of the proportions of male within each age class (example (i) above), then the subcommand `categories` for the `@observation[label].type=proportion_by_age` command is,

```
categories male
```

SPM then expects that there will be a vector of proportions supplied, with one proportion for each age class within the defined age range. For example, if the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10). The expected values will be the expected proportions of males within each of these age classes, after applying a selectivity at the year and time step specified. Note that the supplied vector of proportions (i.e., in this example, the 8 proportions) must sum to one (with a default tolerance of 0.001).

By default, categories are aggregated. For example, to specify that the observations are of the proportions of total individuals (males + females) at each age class (example (ii) above), then the subcommand `categories` is,

```
categories male female
```

`SPM` then expects that there will be a vector of proportions supplied, with one proportion for each age class within the defined age range. For example, if the age range was 3 to 10, then 8 proportions should be supplied. The expected values will be the expected proportions of selected males plus selected females within each of these age classes, at the year and time step specified. Note that the supplied vector of proportions (i.e., in this example, the 8 proportions) must sum to one (with a default tolerance of 0.001).

Otherwise, to provide proportions for multiple categories simultaneously, you need to specify which categories are aggregated and which are separate. `SPM` uses a : symbol to denote those categories to separate. For example, to specify that the observations are of the the proportions of individuals for both male and female simultaneously at each age class (example (iii) above), then the subcommand `categories` is,

```
categories male : female
```

`SPM` then expects that there will be a vector of proportions supplied, with one proportion for each age class within the defined age range for males and then females. For example, if the age range was 3 to 10, then 16 proportions should be supplied, the first set of 8 corresponding to the proportions of male (out of male and female combined) and the second 8 to female. The expected values will be the expected proportions of selected males and selected females within each of these age classes, at the year and time step specified. Note that the supplied vector of proportions (i.e., in this example, the 16 proportions) must sum to one (with a default tolerance of 0.001).

The definition of categories can be a combination of the above cases and can become more complex as the number of categories in the model increases. For example, in a model with categories male-immature, male-mature, female-immature, and female-mature, we might supply observations of males (immature and mature combined) and females (immature and mature combined) simultaneously. Here the `categories` subcommand would be;

```
categories male-immature male-mature : female-immature female-mature
```

Assuming that we wanted to supply observations for ages 3 to 10, then `SPM` would expect a vector of 16 proportions for each spatial location — the first eight corresponding to the observed proportions of males aged 3–10, and the second set of eight corresponding to the proportions of females aged 3–10. Further, the proportions would sum to one.

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

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

```
@layer Area
```

```
type categorical
data A A
data B B
```

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

```
@observation MyProportions
...
categories male female
min_age 1
max_age 5
obs A 0.01 0.05 0.10 0.20 0.20 0.01 0.05 0.15 0.20 0.03
...
```

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

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

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

SPM always evaluates the observations at the end of a time step (i.e., after SPM has applied all of the processes for that time step). However, the observation can be applied to the abundance at the start of a time step or part-way through a time step by the use of the proportion_time_step subcommand. Here SPM stores the state of the partition at the beginning of a time step, and again at the end of the time step. The partition at some point p during the time step is then evaluated as the weighted sum between the start and end of the time step, i.e, for any element i in the partition, $n_i = (1-p)n_i^{start} + pn_i^{end}$.

6.2.1 Likelihoods for proportions-at-age observations

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

The multinomial likelihood

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

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(Z(E_i, \delta)) \tag{6.1}$$

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

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

The default value of δ is 1×10^{-11}.

The lognormal likelihood

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

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i / Z(E_i, \delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{6.3}$$

where

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

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.5}$$

The default value of δ is 1×10^{-11}.

6.3 Proportions-by-category

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

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

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

Proportions-by-category observations can be supplied for any set of categories as a proportion of themselves and any set of additional categories. For example, for a model with the two categories *male* and *female*, we might supply observations of the proportions of males in the population at each

age class. The subcommand `categories` defines the categories for the numerator in the calculation of the proportion, and the subcommand `categories2` supplies the additional categories to be used in the denominator of the calculation. In addition, each category must have an associated selectivity, defined by `selectivities` for the numerator categories and `selectivities2` for the additional categories used in the denominator, e.g.,

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

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

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

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

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

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

```
@observation MyProportions
...
categories male
categories2 female
min_age 1
max_age 5
obs A 0.01 0.05 0.10 0.20 0.20 0.01 0.05 0.15 0.20 0.03
...
```

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

```
@observation MyProportions
...
categories male
categories2 female
min_age 1
max_age 5
```

```
obs A 0.01 0.05 0.10 0.20 0.20 0.01 0.05 0.15 0.20 0.03
obs B 0.02 0.06 0.10 0.21 0.18 0.02 0.05 0.15 0.20 0.01
...
```

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

SPM always evaluates the observations at the end of a time step (i.e., after SPM has applied all of the processes for that time step). However, the observation can be applied to the abundance at the start of a time step or part-way through a time step by the use of the proportion_time_step subcommand. Here SPM stores the state of the partition at the beginning of a time step, and again at the end of the time step. The partition at some point p during the time step is then evaluated as the weighted sum between the start and end of the time step, i.e, for any element i in the partition, $n_i = (1-p)n_i^{start} + pn_i^{end}$.

6.3.1 Likelihoods for proportions-by-category observations

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

The binomial likelihood

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

$$\begin{aligned} -\log(L) = -\sum_i \log(N_i!) - \log((N_i(1-O_i))!) - \log((N_i O_i)!) + N_i O_i \log(Z(E_i,\delta)) \\ + N_i(1-O_i)\log(Z(1-E_i,\delta)) \end{aligned} \tag{6.6}$$

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

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

The default value of δ is 1×10^{-11}.

The normal approximation to the binomial likelihood

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

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{6.8}$$

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

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

The default value of δ is 1×10^{-11}.

6.4 Abundance

Abundance observations are observations of either a relative or absolute number of individuals from a set of categories after applying a selectivity.

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

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

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

defines an observation of the abundance of males. `SPM` then expects that there will be a single abundance value supplied. The expected values will be the expected number of males, after applying the selectivities, at the year and time step specified.

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

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

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

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

```
@observation MyAbundance
...
categories male
obs A 1000
...
```

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

```
@observation MyProportions
...
categories male
obs A 1000
obs B 1200
...
```

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

SPM always evaluates the observations at the end of a time step (i.e., after SPM has applied all of the processes for that time step). However, the observation can be applied to the abundance at the start of a time step or part-way through a time step by the use of the proportion_time_step subcommand. Here SPM stores the state of the partition at the beginning of a time step, and again at the end of the time step. The partition at some point p during the time step is then evaluated as the weighted sum between the start and end of the time step, i.e, for any element i in the partition, $n_i = (1-p)n_i^{start} + pn_i^{end}$.

6.4.1 Likelihoods for abundance observations

The lognormal likelihood

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

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i/qZ(E_i,\delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{6.10}$$

where

$$\sigma_i = \sqrt{\log\left(1+c_i^2\right)} \tag{6.11}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.12}$$

The default value of δ is 1×10^{-11}.

The normal likelihood

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

$$-\log(L) = \sum_i \left(\log(c_i E_i) + 0.5 \left(\frac{O_i - E_i}{Z(c_i E_i, \delta)} \right)^2 \right) \tag{6.13}$$

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

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

The default value of δ is 1×10^{-11}.

6.5 Process error

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

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

$$c_i' = \sqrt{c_i^2 + c_{process\_error}^2} \tag{6.15}$$

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

Similarly, if the likelihood is parameterised by the effective sample size N,

$$N_i' = \frac{1}{1/N_i + 1/N_{process\_error}} \tag{6.16}$$

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

For both the c.v. and N process errors, the process error has more effect on small errors than on large ones. Be clear that a large value for the N process error means a small process error.

6.6 Ageing error

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

Ageing error is optional, and if it is used, it may be omitted for any individual time series. Different ageing error models may be applied for different observation commands. See Section 7.12 for reporting the misclassification matrix.

The ageing error models implemented are;

1. None The default model is to apply no ageing error.
2. Off by one Proportion p_1 of individuals of each age a are misclassified as age $a-1$ and proportion p_2 are misclassified as age $a+1$. Individuals of age $a < k$ are not misclassified. If there is no plus group in the population model, then proportion p_2 fish of the oldest age class will 'fall off the edge' and disappear.

3. Normal Individuals of age a are classified as ages which are normally distributed with mean a and constant c.v. c. As above, if there is no plus group in the population model, some individuals of the older age classes may disappear. If c is high enough, some of the younger age classes may 'fall off the other edge'. Individuals of age $a < k$ are not misclassified.

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

6.7 Simulating observations

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

`SPM` generates simulated observations for the observations defined in the input configuration file with `@observation[label].simulate=true`. Note that observations with `@Observation[label].likelihood=none` cannot be used to simulate observations.

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

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

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

(a) A sample of N_i independent binary variates is generated, equalling 1 with probability E_i

(b) The simulated observation value S_i is calculated as the sum of these binary variates divided by N_i

Note that SPM can report simulated observations using the usual observation report (@report[label].type=observation), but that when simulating the contribution to the objective function of each observation is reported as zero. The report @report[label].type=observation_definition will print the complete observation definition in a form suitable for use in a SPM input configuration file. See Section 7 for more detail.

6.8 Pseudo-observations

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

- Output will only be generated if a report command @report.type=observation is specified.
- The observed values should be supplied (even if they are 'dummy' observation). These will be processed by SPM as if they were actual observation values, and must conform to the validations carried out for the other types of likelihood.
- The subcommands likelihood, obs, error_value and process_error have no effect when generating the expected values for the pseudo-observation.
- SPM cannot simulate from pseudo-observations.

7 The report section

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

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

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

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

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

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

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

Note that reports can be defined that are not valid, for example printing the partition for a year and/or time-step that does not exist or reporting the covariance matrix when not estimating — although all reports must conform to syntax requirements. In these cases, SPM does not generate output.

7.1 Printing the model spatial map

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

7.2 Printing the partition

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

7.3 Printing the partition at the end of an initialisation

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

partition at then end of that initialisation phase.

7.4 Printing a process summary

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

7.5 Printing derived quantities

Print out the a description of a derived parameter, and its values of a derived quantity as recorded in the model state, for each year of the model. In addition, the report prints out the values of the derived quantity at the end of each initialisation phase.

7.6 Printing the estimated parameters

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

7.7 Printing the estimated parameters in a vector format

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

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

7.8 Printing the objective function

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

7.9 Printing the covariance matrix

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

7.10 Printing observations, fits, and residuals

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

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

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

7.11 Printing simulated observations

Prints out a complete observation definition in a form suitable for use in a `SPM` input configuration file, reproducing the command and subcommands from the input configuration file, but with the observations replaced with simulated values.

7.12 Printing the ageing error misclassification matrix

Prints out the ageing error misclassification matrix.

7.13 Printing layers

Prints the values in the layer (including user supplied layers, abundance and biomass layers, and meta-layers) for a given year and at the end of a given time step.

7.14 Printing a derived view via a categorical layer

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

7.15 Printing selectivities

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

7.16 Printing the random number seed

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

7.17 Verifying the size-weight relationship

Prints a summary of the size-weight relationship used by `SPM` for each category. Useful for verifying that the choice of parameters (and their magnitude) are sensible choices.

8 Population command and subcommand syntax

8.1 Model structure

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

`nrows` The number of rows n_{rows} in the spatial structure
Type: Integer
Default: 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 $1 \ldots n_{categories}$
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 $1 \ldots n_{categories}$
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, ≥ 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. 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: 1.0
Value: Steepness value between 0.2 and 1.0

`sigma_r` Define the recruitment variability σ_R in the stock-recruitment relationship for projections
Type: Estimable
Default: 1.0

`rho` Define the autocorrelation ρ in the recruitment variability in the stock-recruitment relationship for projections
Type: Estimable
Default: 0.0

`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 ≥ 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: No layer
Value: A valid layer as defined by `@layer`. If a numeric layer, then recruitment is in proportion to the layer values.

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 ≥ 0 and ≤ 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: No layer
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. 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 ≥ 0 and ≤ 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: No layer
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. 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: 0.99
Value: Must be > 0 and < 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: 0.99
Value: Must be > 0 and < 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`

`proportions` Define the proportion of individuals to move
Type: Estimable
Default: None
Value: A value ≥ 0 and ≤ 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. 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.

`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

`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 α
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 α
Type: Estimable
Default: None

`mu` Defines the μ parameter of the normal preference function
Type: Estimable
Default: None

`sigma` Defines the σ 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 α
Type: Estimable
Default: None

`mu` Defines the μ parameter of the double-normal preference function
Type: Estimable
Default: None

`sigma_l` Defines the σ_L parameter of the double-normal preference function
Type: Estimable
Default: None

`sigma_r` Defines the σ_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 α
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 α

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 α
Type: Estimable
Default: None

`lambda` Defines the λ 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 α

Type: Estimable
Default: None

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

`lambda` Defines the λ 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 α
Type: Estimable
Default: None

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

`lambda` Defines the λ 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` × `@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` × `@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`

`time_step_proportion` Define the proportion of the time step through which the derived quantity has been calculated
Type: String vector
Default: None
Value: Valid selectivity labels from `@selectivity`

`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.

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.

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_∞ 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 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, ≤ 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 τ_1 parameter of the Schnute relationship
Type: Estimable
Default: None
Value: A real number

`tau2` Define the τ_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 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, ≤ 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 α 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 α parameter of the selectivity function
Type: Estimable
Default: None
Value: A positive real number

`a50` 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

8.10.7 `@selectivity[label].type=logistic_producing`

`alpha` Defines the α 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

`a50` 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 α parameter of the selectivity function
Type: Estimable
Default: None
Value: A positive real number

`mu` Defines the μ parameter of the selectivity function
Type: Estimable
Default: None

`sigma_l` Defines the σ_L parameter of the selectivity function
Type: Estimable
Default: None

`sigma_r` Defines the σ_R parameter of the selectivity function
Type: Estimable
Default: None

8.10.9 @selectivity[label].type=double_exponential

`alpha` Defines the α 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

`y0` Defines the y_0 parameter of the selectivity function
Type: Estimable
Default: None

`y1` Defines the y_1 parameter of the selectivity function
Type: Estimable
Default: None

`y2` 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

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

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

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

`tolerance` Define the convergence criteria (tolerance) for the minimiser
Type: Constant
Default: 0.002
Value: A positive real number

`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating
Type: Logical
Default: True
Value: Either true of false

9.2.2 @minimiser[label].type=DE_solver

`population_size` Define the minimisers number of populations to generate
Type: Integer
Default: 25
Value: A positive integer

`crossover_probability` Define the minimisers crossover probability
Type: Integer
Default: 0.9
Value: A positive integer

`difference_scale` Define the scale of the difference of the parent candidates for the minimiser

Type: Constant
Default: 0.02
Value: A positive real number

`max_generations` Define the maximum generations for the minimiser convergence
Type: Constant
Default: 0.002
Value: A positive real number

`tolerance` Define the convergence criteria (tolerance) for the minimiser
Type: Constant
Default: 0.01
Value: A positive real number

`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating
Type: Logical
Default: True
Value: Either true of false

9.3 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 ¿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 -max_correlation are increased to -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 μ 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 μ 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 σ 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 μ 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 μ of the Beta prior
Type: Constant
Default: No default
Value: Defines the mean of the Beta prior

`sd` Defines the standard deviation σ of the Beta prior
Type: Constant
Default: No default
Value: Defines the standard deviation of the Beta prior

9.7 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

9.8 Defining penalties

`@penalty` *`label`* Define a penalty with `label`

`log_scale` Defines if the penalty in calculated in log space
Type: Switch
Default: False
Value: Defines if the value of the penalty is calculated as the squared difference or the squared difference in log space

`multiplier` Penalty multiplier
Type: Constant
Default: 1.0
Value: Defines the penalty multiplier

10 Observation command and subcommand syntax

10.1 Observation types

The observation types available are,

Observations of 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: True
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: No process error
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: True
Value: Either true or false

`ageing_error` Define the label of the ageing-error matrix to be applied
Type: String
Default: None
Value: A valid ageing error label

`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: No process error
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: True
Value: Either true or false

`ageing_error` Define the label of the ageing-error matrix to be applied
Type: String
Default: None
Value: A valid ageing error label

`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: No process error
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: No process error
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

10.2 Defining ageing error

Three methods for including ageing error into estimation with observations are,

- None
- Normal
- Off-by-one

Each type of ageing error requires a set of subcommands and arguments specific to its type.

`@ageing_error` *`label`* Define ageing error with `label`

`type` The type of ageing error
Type: String
Default: None
Value: Defines the type of ageing error to use

10.2.1 `@ageing_error[label].type=none`

The `@ageing_error[label].type=none` has no other subcommands.

10.2.2 @ageing_error[label].type=normal

`cv` Parameter of the normal ageing error model
Type: Constant
Default: None
Value: Define the c.v. of misclassification

`k` The k parameter of the normal ageing error model
Type: Integer
Default: 0
Value: cv defines the proportions of misclassification down and up using the normal model. k defines the minimum age of fish which can be misclassified — fish under age k have no ageing error

10.2.3 @ageing_error[label].type=off_by_one

`p1` The p_1 parameter of the off-by-one ageing error model
Type: Constant
Default: None
Value: p_1 and p_2 define the proportions of misclassification 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

`p2` The p_2 parameter of the off-by-one ageing error model
Type: Constant
Default: None
Value: p_1 and p_2 define the proportions of misclassification 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

`k` The k parameter of the off-by-one ageing error model
Type: Integer
Default: 0
Value: p_1 and p_2 define the proportions of misclassification down and up by 1 year respectively. k defines the minimum age of fish which can be misclassified — fish under age k have no ageing error

11 Report command and subcommand syntax

11.1 Reports

The report types available are,

1. Print the map (i.e., row and column labels of each spatial cell) of the spatial structure
2. Print the partition for a year and time step
3. Print the partition at the end of an initialisation
4. Print a summary of a process
5. Print a derived quantity
6. Print a summary of the estimated parameters
7. Print the estimated parameters in a vector format (suitable for use with `spm -i`)
8. Print the objective function values
9. Print the covariance matrix
10. Print an observation values, fits, and residuals
11. Print a simulated observation definition suitable for use in a `SPM` input configuration file.
12. Print the ageing error misclassification matrix
13. Print a layer
14. Print a derived view via a categorical layer
15. Print a selectivity's values
16. Print the random number seed
17. 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=spatial_map`

`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

11.1.2 @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 of False

11.1.3 @report[label].type=initialisation

`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 of False

11.1.4 @report[label].type=process

`process` Define the label of the process to summarise
Type: String
Default: None
Value: A valid label from `@process`

`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 of False

11.1.5 @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 of False

11.1.6 @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 of False

11.1.7 @report[label].type=estimate_value

Prints the estimated parameters in a format suitable for use with `spm -i`.

`file_name` Define the name of the output file where the report is written
Type: String
Default: 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 of False

11.1.8 @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 of False

11.1.9 @report[label].type=covariance

`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 of False

11.1.10 @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 of False

11.1.11 @report[label].type=observation_definition

`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 of False

11.1.12 @report[label].type=ageing_error

`ageing_error` Define the label of the ageing_error misclassification matrix
Type: String
Default: None
Value: A valid label from `@ageing_error`

`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 of False

11.1.13 @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 of False

11.1.14 @report[label].type=layer_derived_view

`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 of False

11.1.15 `@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 of False

11.1.16 @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 of False

11.1.17 @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 of 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

A set of examples are provided to demonstrate syntax of the input configuration file, provide an introduction to the command calls to SPM, and respective outputs. In each of the examples below, we reproduce only a subset of the input configuration files — for more detail, see the example files.

13.1 An example of a simple 1×1 non-spatial model

The first example implements a very simple single spatial cell model (i.e., no movement 1×1 spatial structure) with recruitment, maturation, natural and exploitation mortality, and an annual age increment. The population structure has ages $1-50^{+}$ with categories labelled immature and mature.

The model is initialised over a 200 year period, and applies the following processes,

1. A constant recruitment process, recruiting a constant number of individuals to the first age class (i.e., $age = 1$) in the category labelled immature.
2. A maturation process, where individuals are moved from the immature to the mature categories with a logistic-producing selectivity labelled 'maturation'.
3. A constant mortality process representing natural mortality, applied as two repeats of the 'halfM' process. (Half M used so-as to be able to mimic a $\frac{1}{2}M + F + \frac{1}{2}M$ natural and fishing exploitation set of processes after initialisation.)
4. An ageing process, where all individuals are aged by one year, and with a plus group accumulator age class at $age = 50$.

A second phase of initialisation, of period one year, is applied to allow external validation that the initialisation process has stabilised the population to equilibrium (i.e., by confirming that there is no or at least a small difference in the partition at the end of first and second phases).

Following initialisation, the model runs from the years 1994 to 2007 iterating through two time steps. The first time step applies processes of constant recruitment, maturation, and $\frac{1}{2}M + F + \frac{1}{2}M$ processes. The exploitation process (fishing) is applied in the years 1998–2007, with catches defined by the layers Fishing_1998 – Fishing_2007.

The second time step applies an age increment.

The first 50 lines of the main section of the input configuration file are,

```
# Model structure
@model
nrows 1
ncols 1
layer Base
categories immature mature
min_age 1
max_age 35
age_plus_group True
initial_year 1994
current_year 2007
final_year 2108
cell_length 1
initialisation_phases Phase1 Phase2
time_steps step_one step_two

# Initalisation
@initialisation_phase Phase1
```

```
years 200
processes Recruitment Maturation halfM halfM Ageing

@initialisation_phase Phase2
years 1
processes Recruitment Maturation halfM halfM Ageing

# Annual cycle
@time_step step_one
processes Recruitment Maturation halfM fishing halfM

@time_step step_two
processes Ageing

# Population processes
@process Ageing
type ageing
categories immature mature

@process Recruitment
type constant_recruitment
categories immature
proportions 1.0
R0 500000
ages 1

@process halfM
type constant_mortality_rate
categories immature mature
M 0.065 0.065
selectivities One One
```

The input configuration file includes definitions of required layers and the estimation, observation, and report parameters as external files.

To carry out a run of the model (to verify that the model runs without any syntax errors), use the command `spm -r -c config.spm`. Note that as `SPM` looks for a file named `config.spm` by default, we can simplify the command to `spm -r`.

To run an estimate, and hence estimate the parameters defined in the file `estimation.spm` (the catchability constant q, recruitment R_0, and the fishing selectivity parameters a_{50} and $a_{to}95$), use `spm -e`. Here, we have piped the output to `estimate.log` using the command `spm -e > estimate.log`. `SPM` reports a the results of each iteration of the estimation, and ends with successful convergence,

```
Convergence was successful
Total elapsed time: 1 second
```

The main part of the output from the estimation run is summarised in the file `estimate.log`, and the final objective function is,

```
[objective_score]
report.type: objective_function
obs->caa-year-1998: 28.1636
obs->caa-year-1999: 33.1757
obs->caa-year-2000: 27.6867
obs->caa-year-2001: 26.694
obs->caa-year-2002: 25.6249
obs->caa-year-2003: 30.5752
obs->caa-year-2004: 31.5709
```

```
obs->caa-year-2005: 28.2699
obs->caa-year-2006: 26.9388
obs->caa-year-2007: 29.9528
obs->cpue-1998: -0.580173
obs->cpue-1999: -0.667389
obs->cpue-2000: -1.37969
obs->cpue-2001: 0.31442
obs->cpue-2002: -0.388211
obs->cpue-2003: -1.20084
obs->cpue-2004: -0.388395
obs->cpue-2005: -1.41054
obs->cpue-2006: -0.490961
obs->cpue-2007: 1.7633
prior->catchability[cpueq].q: 0
prior->process[recruitment].r0: 0
prior->selectivity[fishingsel].a50: 0
prior->selectivity[fishingsel].ato95: 0
total_score: 284.224
*end
```

with parameter estimates,

```
[estimate-values]
report.type: estimate_summary
parameter: catchability[cpueq].q
lower_bound: 1e-010
upper_bound: 0.1
prior: uniform
value: 0.0001

parameter: process[recruitment].r0
lower_bound: 10000
upper_bound: 1e+007
prior: uniform
value: 500000

parameter: selectivity[fishingsel].a50
lower_bound: 1
upper_bound: 20
prior: uniform
value: 8

parameter: selectivity[fishingsel].ato95
lower_bound: 0.01
upper_bound: 50
prior: uniform
value: 3

*end
```

A profile on the R_0 parameter can also be run, using `spm -p > profile.log`.

13.2 An example of a simple 10×10 spatial model

```
max_age 30
age_plus_group True
initialisation_phases Phase1 Phase2 Phase3
initial_year 1995
Current_year 2007
final_year 2107
```

```
cell_length 100
time_steps one two

# Initialisation
@initialisation_phase Phase1
years 100
processes Recruitment Maturation halfM halfM Ageing

@initialisation_phase Phase2
years 100
processes Recruitment Maturation halfM halfM MoveImmature MoveMature Ageing

@initialisation_phase Phase3
years 1
processes Recruitment Maturation halfM halfM MoveImmature MoveMature Ageing

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

@time_step two # Winter
processes MoveImmature MoveMature Ageing

# Derived quantities
@derived_quantity SSB
time_step one
categories mature
```

13.3 An example of a more complex 10×10 spatial model

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 you own analysis functions. See the 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 the 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 v1.00-2009-05-21 (rev. 3374) 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 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 the movement paradigm employed in 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 some of the definitions, 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 σ_R in the stock-recruitment relationship for projections
rho Define the autocorrelation ρ 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
`proportions` 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`

`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 α

@preference_function[label].type=normal

layer Defines the layer which supplies the preference function independent variable
alpha Defines the multiplicative constant α
mu Defines the μ parameter of the normal preference function
sigma Defines the σ parameter of the normal preference function

@preference_function[label].type=double_normal

layer Defines the layer which supplies the preference function independent variable
alpha Defines the multiplicative constant α
mu Defines the μ parameter of the double-normal preference function
sigma_l Defines the σ_L parameter of the double-normal preference function
sigma_r Defines the σ_R parameter of the double-normal preference function

@preference_function[label].type=logistic

layer Defines the layer which supplies the preference function independent variable
alpha Defines the multiplicative constant α
a50 Defines the a_{50} parameter of the logistic preference function
ato95 Defines the a_{to95} parameter of the logistic preference function

@preference_function[label].type=inverse_logistic

layer Defines the layer which supplies the preference function independent variable
alpha Defines the multiplicative constant α
a50 Defines the a_{50} parameter of the inverse-logistic preference function
ato95 Defines the a_{to95} parameter of the inverse-logistic preference function

@preference_function[label].type=exponential

layer Defines the layer which supplies the preference function independent variable
alpha Defines the multiplicative constant α
lambda Defines the λ parameter of the exponential preference function

@preference_function[label].type=threshold

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

@preference_function[label].type=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 α
`biomass` Defines the B biomass parameter of the threshold biomass preference function
`lambda` Defines the λ 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
time_step_proportion Define the proportion of the time step through which the derived quantity has been 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_∞ 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 τ_1 parameter of the Schnute relationship
tau2 Define the τ_2 parameter of the Schnute relationship
a Define the a parameter of the Schnute relationship
b Define the b parameter of the Schnute relationship
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 α parameter of the selectivity function
l Defines the *L* parameter of the selectivity function
h Defines the *H* parameter of the selectivity function
v Defines the *V* parameters (one for each age class from *L* to *H*) of the selectivity function

@selectivity[label].type=logistic

alpha Defines the α parameter of the selectivity function
a50 Defines the a_{50} parameter of the selectivity function
ato95 Defines the a_{to95} parameter of the selectivity function

@selectivity[label].type=logistic_producing

alpha Defines the α parameter of the selectivity function
l Defines the *L* parameter of the selectivity function
h Defines the *H* parameter of the selectivity function
a50 Defines the a_{50} parameter of the selectivity function
ato95 Defines the a_{to95} parameter of the selectivity function

@selectivity[label].type=double_normal

alpha Defines the α parameter of the selectivity function

`mu` Defines the μ parameter of the selectivity function
`sigma_l` Defines the σ_L parameter of the selectivity function
`sigma_r` Defines the σ_R parameter of the selectivity function

`@selectivity[label].type=double_exponential`

`alpha` Defines the α parameter of the selectivity function
`x1` Defines the x_1 parameter of the selectivity function
`x2` Defines the x_2 parameter of the selectivity function
`x0` Defines the x_0 parameter of the selectivity function
`y0` Defines the y_0 parameter of the selectivity function
`y1` Defines the y_1 parameter of the selectivity function
`y2` Defines the y_2 parameter of the selectivity function

`@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`

`iterations` Define the maximum number of iterations for the minimiser
`evaluations` Define the maximum number of evaluations for the minimiser
`step_size` Define the step-size for the minimiser
`tolerance` Define the convergence criteria (tolerance) for the minimiser
`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating

`@minimiser[label].type=DE_solver`

`population_size` Define the minimisers number of populations to generate
`crossover_probability` Define the minimisers crossover probability
`difference_scale` Define the scale of the difference of the parent candidates for the minimiser
`max_generations` Define the maximum generations for the minimiser convergence
`tolerance` Define the convergence criteria (tolerance) for the minimiser
`covariance` Specify if `SPM` should attempt to calculate the covariance matrix, if estimating

`@MCMC` *`label`* Define the MCMC estimation arguments

`type` Define the method of MCMC

`@MCMC.type=Metropolis_Hastings`

`start` Covariance multiplier for the starting point of the Markov chain
`length` Length of the Markov chain
`keep` Spacing between recorded values in the chain
`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
`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 μ of the normal prior
`cv` Defines the c.v. c of the normal prior

`@prior[label].type=normal_by_sd`

`mu` Defines the mean μ of the normal by standard deviation prior
`sd` Defines the standard deviation σ of the normal by standard deviation prior

`@prior[label].type=lognormal`

`mu` Defines the mean μ 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 μ of the Beta prior
`sd` Defines the standard deviation σ of the Beta prior

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

`q` Value of the q parameter

`@penalty` *`label`* Define a penalty with `label`

`log_scale` Defines if the penalty in calculated in log space
`multiplier` Penalty multiplier

17.3 Observation command and subcommand syntax

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

`type` Define the type of observation

`@observation[label].type=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
`ageing_error` Define the label of the ageing-error matrix to be applied
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`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
`ageing_error` Define the label of the ageing-error matrix to be applied
`layer` Name of the categorical layer used to group the spacial cells for the observation
`obs [label]` Define the following data as observations for the categorical layer with value `[label]`
`error_value [label]` Define the following data as error values (e.g., N for multinomial likelihoods, c.v. for lognormal likelihoods, etc.) for the categorical layer with value `[label]`
`likelihood` Define the likelihood for the observation
`delta` Define the delta robustifying constant for the likelihood
`process_error` Define the process error term
`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

`@ageing_error` *`label`* Define ageing error with `label`

`type` The type of ageing error

`@ageing_error[label].type=none`

`@ageing_error[label].type=normal`

`cv` Parameter of the normal ageing error model
`k` The k parameter of the normal ageing error model

`@ageing_error[label].type=off_by_one`

`p1` The p_1 parameter of the off-by-one ageing error model
`p2` The p_2 parameter of the off-by-one ageing error model
`k` The k parameter of the off-by-one ageing error model

17.4 Report command and subcommand syntax

@report *label* Define an output report

type Define the type of report

@report[label].type=spatial_map

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

@report[label].type=partition

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

initialisation_phase Define the phase of initialisation that the partition report applies to
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=process

process Define the label of the process to summarise
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=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_summary

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=estimate_value

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=objective_function

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=covariance

file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=observation

observation Define the label of the observation to print
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=observation_definition

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=ageing_error

ageing_error Define the label of the ageing_error misclassification matrix
file_name Define the name of the output file where the report is written
overwrite Specify if any previous file with the same name as the output file should be overwritten or appended to

@report[label].type=layer

layer Define the label of the layer to print
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=layer_derived_view

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

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