Casal2 User Manual

S. Rasmussen, I. Doonan, A. Dunn, C. Marsh, K. Large, S. Mormede

DRAFT

NIWA Technical Report xxx
ISSN 1174-2631
2016

Casal2 User Manual (modified: 2016-05-22)
for use with Casal2 2016-05-22 (rev. 2271ffd)

Citation: S. Rasmussen, I. Doonan, A. Dunn, C. Marsh, K. Large, S. Mormede (2016) Casal2 User Manual, 2016-05-22 (rev. 2271ffd). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report*. 203 p.

Contents

1. Introduction

Casal2 (`casal2`) is a generalised age- or length-structured fish stock assessment model that allows flexibility in specifying population dynamics, paramaeter estimation and model outputs. Casal2 can model population dynamics for an age- or length-structured population using a range of observations, including tagging, relative abundance, and age frequency data. Casal2 implements an age-structured population which can have user defined categories (e.g., immature, mature, male, female, predator, prey etc.), and age range.

This manual describes how to use Casal2, including how to run Casal2, how to set up an input configuration file. Further, we describe the population dynamics and estimation methods, and describe how to specify and interpret output.

1.1. Version

This document (last modified 2016-05-22) describes Casal2 2016-05-22 (rev. 2271ffd). The Casal2 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 Casal2, and can be obtained, on request, from the authors.

1.2. Citing Casal2

A suitable reference for Casal2 and this document is:

S. Rasmussen, I. Doonan, A. Dunn, C. Marsh, K. Large, S. Mormede (2016) Casal2 User Manual, 2016-05-22 (rev. 2271ffd). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report*. 203 p.

1.3. Software license

This program and the accompanying materials are made available under the terms of the licence GNU GPL v2 which accompanies this software (see Section 19).

Copyright ©2008-2016, National Institute of Water & Atmospheric Research Ltd. and the New Zealand Ministry for Primary Industries. All rights reserved.

1.4. System requirements

Casal2 is available for most IBM compatible machines running 64-bit Linux and Microsoft Windows operating systems.

Several of Casal2s tasks are highly computer intensive and a fast processor is recommended. Depending on the model implemented, some of Casal2s tasks can take a considerable amount of time (minutes to hours), and in extreme cases can even take several days to estimate a model fit. Multi-core machines are necessary when running Casal2

The program itself requires only a few megabytes of hard-disk space but output files can consume large amounts of disk space. Depending on number and type of user output requests, the output could range from a few hundred kilobytes to several hundred megabytes. When estimating model fits, several hundred megabytes of RAM may be required, depending on the spatial size of the model, number of categories, and complexity of processes and observations. For extremely large models, several gigabytes of RAM may be required.

1.5. Necessary files

For both 64-bit Linux and Microsoft Windows, only the binary file `casal2` or `casal2.exe` is required to run Casal2. No other software is required. We do not compile a version for 32-bit operating systems.

Casal2 offers little in the way of post-processing of the output, and a package available that allows tabulation and graphing of model outputs is recommended. We suggest software such as **R** (R Development Core Team 2007) to assist in the post processing of Casal2 output. We provide the `CASAL2` **R** package for importing the Casal2 output into **R** (see Section 14).

1.6. Getting help

Casal2 is distributed as unsupported software, however we would appreciate being notified of any problems or errors in Casal2. See Section 15.2 for how to report errors, for reporting errors and further information on Casal2 contact the development team at casal2@niwa.co.nz.

1.7. Technical details

Casal2 was compiled on Linux using `gcc`, the C/C++ compiler developed by the GNU Project. The 64-bit Linux version was compiled using `gcc` version 5.2.1 20151010 (Ubuntu Linux). The Microsoft Windows version was compiled using Mingw32 `gcc` (tdm64-1) 5.1.0. The Microsoft Windows installer was built using the Inno Setup 5 application.

Casal2 six 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 third and last non auto differential algorithm is the Dlib (King, 2009). ADOLC is an auto differential minimiser more information can be found at Walther et al. (1996), CPPAD is another auto differential minimiser that can be used in Casal2 and more information can be found at Wächter and Biegler (2006) and BETADIFF which is a modified version of ADOL-C v1.8.4 by Brian Bull, and is the only auto differential library form the predecessor CASAL.

The random number generator used by Casal2 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 1.58.0).

Note that the output from Casal2 may differ slightly on the different platforms due to different precision arithmetic or other platform dependent implementation issues. The source code for Casal2 is available in the windows bundle or on the github repository at `https://github.com/alistairdunn1/CASAL2`

Unit tests of the underlying Casal2 code are carried out at build time, using the GOOGLE mock and unit testing framework. The unit test framework aims to cover a significant proportion of the key functionality within the Casal2 code base. The unit test code for Casal2 is available as a part of the underlying source code.

2. Model overview

2.1. Introduction

Casal2 is an age-structured population dynamics model. It implements a statistical catch-at-age population dynamics, using a discrete time-step state-space model that represents a cohort-based population age structure .

Casal2 is run from the console window on Microsoft Windows or from a terminal window on Linux. Casal2gets its information from input data files, the main one of which is the *input configuration file*. Commands and subcommands in the input configuration file are used to define the model structure, provide observations, define parameters, and define the outputs (reports) for Casal2. Command line switches tell Casal2 the run mode and where to direct its output. See Section 3 for the details.

We define the model in terms of the *state*. The state consists of two parts, the *partition*, and any *derived quantities* or *derived quantities by cell*. The state will typically change one or more times in every *time-step* of every year, depending on the *processes* defined for each model.

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

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

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

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

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

The population structure of Casal2 follows the usual population modelling conventions and is similar to those implemented in other population models, for example CASAL (Bull et al., 2012). The model records the numbers of individuals by age and category (e.g., male, female). 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 Casal2 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. Casal2 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* into the future or used by Casal2 as parameters of an operating model to *simulate* observations.

A model in Casal2 is specified by an input configuration file, and comprises of four main components. These are the population section (model structure, population 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 Casal2. See Sections 8, 9, 10, and 11 for details and specification of Casal2s command and subcommand syntax within the input configuration file.

2.2. The population section

The population section (Section 4) defines the model of the population dynamics. It describes the model structure (i.e. the population structure), initialisation, run and projection years (model period), population processes (for example, recruitment, migration, and mortality), 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, tag recapture observations, 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 Casal2 will provide informational messages to the screen, Casal2 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 Casal2

Casal2 is run from the console window (i.e., the DOS command line) on Microsoft Windows or from a terminal window on Linux. Casal2 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 Casal2 configuration file. By convention, the name of the input configuration file ends with the suffix `.csl2`, however, any file name is acceptable.

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

Simple command line arguments are used to determine the actions or *tasks* of Casal2, 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 Casal2

To use Casal2, 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 `casal2` with any arguments (see Section 3.4 for the the list of possible arguments). Casal2 will print output to the screen and return you to the command prompt when it completes its task. Note that the Casal2 executable (binary) and shared libraries (extension `.dll`) must be either in the directory where you run it or somewhere in your systems `PATH`. We are currently working on an installer which will be available soon. 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 Casal2 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 (especially when you have a model that has many observations), and can result in a input configuration file that is long and difficult to navigate. To aid readability and flexibility, we can use the input configuration file command !*`include file`*. The command causes an external file, *`file`*, to be read and processed, exactly as if its contents had been inserted in the main input configuration file at that point. The file name must be a complete file name with extension, but can use either a relative or absolute path as part of its name. Note that included files can also contain !*`include`* commands — but be careful that you do not set up a recursive state. See Section 12 for more detail.

3.3. Redirecting standard output

Casal2 uses the standard output stream `standard output` to display run-time information. The standard error stream is used by Casal2 to output the program exit status and run-time errors. We suggest redirecting both the standard output and standard error into files. With the bash shell (on Linux systems), you can do this using the command structure,

```
(casal2 [arguments] > out) >& err &
```

It may be useful to redirect the standard input, especially if you're using Casal2 inside a batch job software, i.e.

```
(casal2 [arguments] > out < /dev/null) >& err &
```

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

```
casal2 [arguments] > out
```

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

```
casal2 [arguments] > out 2> err
```

Note that Casal2 outputs a few lines of header information to the output. The header consists of the program name and version, the arguments passed to Casal2 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 Casal2 used to run the model.

3.4. Command line arguments

The call to Casal2 is of the following form:

`casal2[-c` *`config_file`*`] [`*`task`*`] [`*`options`*`]`

-c *config_file* Define the input configuration file for Casal2. If omitted, then Casal2 looks for a file named `casal2.txt`.

and where *task* is one of;

-h Display help (this page).

-l Display the reference for the software license (GPL v2).

-v Display the Casal2 version number.

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

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

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

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

-f Project the model *forward* in time using the parameter values in the input configuration file as the starting point for the estimation, or optionally, with the start values from the file denoted with the command line argument `-i` *`file`*.

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

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

-i file *Input* one or more sets of free (estimated) parameter values from *`file`*. See Section 11 for details about the format of *`file`*.

-o file *Output* a report of the free (estimated) parameter values in a format suitable for `-i` *`file`*. See Section 11 for details about the format of *`file`*.

-g seed Seed the random number *generator* with *`seed`*, a positive (long) integer value. Note, if `-g` is not specified, then Casal2 will generate a random number seed based on the computer clock time.

--loglevel arg = {trace, finest, fine, medium} See Section 7.

--tabular Run with `-r` or `-f` command it will print `@report` in tabular format. See Section 7.

--single-step Run with `-r`, this additional option will pause the model and ask the user to specify parameters and their values to use for the next iteration. See Section 4.6.1.

3.5. Constructing a Casal2 input configuration files

The model definition, parameters, observations, and reports are specified in an input configuration files. 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

Casal2 has a range of commands that define the model structure, processes, observations, and how tasks are carried out. There are three types of commands,

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

Commands that have a label must have a unique label, i.e., the label cannot be used on more than one command of that type. The labels can contain alpha numeric characters, period ('.'), underscore ('_') and dash ('-')). Labels must not contain white-space, or other characters that are not letters, numbers, dash, period or an underscore. For example,

```
@process BH_Recruitment
or
!include MyModel
```

3.5.2. Subcommands

Subcommands in Casal2 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 12, 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. Casal2 may report an error if they are not supplied in this way, however, in some circumstances a different order may result in a valid, but unintended set of actions, leading to possible errors in your expected results.

The arguments for a subcommand are either:

switch	true/false
integer	an integer number,
integer vector	a vector of integer numbers,
integer range	a range of integer numbers separated by a hyphen (-), e.g. 1994-1996 2000 is expanded to an integer vector of values 1994 1995 1996 2000),
constant	a real number (i.e. double),
constant vector	a vector of real numbers (i.e. vector of doubles),
estimable	a real number that can be estimated (i.e. estimable double),
estimable vector	a vector of real numbers that can be estimated (i.e. vector of estimable doubles),
string	a categorical (string) value, or
string vector	a vector of categorical values.

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

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

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

Estimable parameters are those parameters that Casal2 can estimate, if requested. If a particular parameter is not being estimated in a particular model run, then it acts as a constant. Within Casal2 only estimable parameters can be estimated. And, you have to tell Casal2 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 unique label or an argument. Each command is then followed by its subcommands and their arguments, e.g.,

`@command`, or

`@command argument`, or

`@command` *`label`*

and then

`subcommand argument`

`subcommand argument`

etc,.

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

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

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

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.)

```
## This is a comment and will be ignored by CASAL2
@process BH_Recruitment
r0 3000000
{
This block of code is a comment and
will also be ignored by CASAL2
}
```

3.5.5. Determining parameter names

When Casal2 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 label. For subcommands, the parameter name format is either

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

`command.subcommand` if the command has no label, or `command[label].subcommand(i)` if the command has a label and the subcommand arguments are a vector, and we are accessing the *i*th element of that vector.

`command[label].subcommand(i:j)` if the command has a label, and the subcommand arguments are a vector, and we are accessing the elements from *i* to *j* (inclusive) of that vector.

The unique parameter name is used to reference the parameter when estimating, applying a penalty, projecting, time varying 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. Casal2 exit status values

When Casal2 completes its task successfully or errors out gracefully, it returns a single exit status value 'completed' to the standard output. Error messages will be printed to the console and if it is a configuration error. It will print the line number and file the error has been identified at

4. The population section

4.1. Introduction

The population section specifies the model structure, population dynamics, and other associated parameters. It describes the model structure (population structure), defines the population processes (e.g., recruitment, migration, and mortality), selectivities, and model parameters.

The population section consists of several components, including;

- The population structure;
- Model initialisation (i.e., the state of the partition at the start of the first year);
- The years over which the model runs (i.e., the start and end years of the model)
- The annual cycle (time-steps and processes that are applied in each time-step);
- The specifications and parameters of the population processes (i.e. processes that add, remove individuals to or from the partition, or shift numbers between ages and categories in the partition);
- Selectivities;
- Parameter values and their definitions;
- Derived quantities, required as parameters for some processes (e.g. Mature biomass to resolve any density dependent processes such as the spawner-recruit relationship, in a recruitment process).

4.2. Population structure

The basic structure of a Casal2 population model is defined in terms of an annual cycle, time steps, states, and transitions.

The annual cycle defines what processes happen in each model year, and in what sequence. Casal2 runs on an annual cycle rather than, for example, a 6-monthly cycle.)

Each year is split into one or more time steps, with at least one process occurring in each time step. Each time step can be thought of as representing a particular part of the calendar year, or you can just treat them as an abstract sequence of events. In every time step, there exists a mortality block, this is a block (a group of consecutive processes) where individuals are removed from the partition. If there are no mortality processes then the mortality block is empty (nothing happens) and occurs at the end of a time step. Casal2 will error out if the user defines multiple mortality processes in a single time step, that are not consecutive processes.

The state is the current status of the population, at any given time. The state can change one or more times in every time step of every year. The state object must contain sufficient information to figure out the future course of the fishery (given a model and a complete set of parameters).

There are a number of possible changes in the state, which are called transitions. These include processes such as recruitment, natural mortality, fishing mortality, ageing, migration, tagging events, and maturation.

The division of the year into an arbitrary number of time steps allows the user to specify the exact order in which processes and observations occur. The user needs to specify the time step in which each process occurs. If more than one process occurs in the same time step, they will be applied in the order specified in the `@time_step` block.

The key element of the state is the partition. This is a broadly applicable concept that can be used to describe many different kinds of population model. The partition is simply a breakdown of the total number of fish in the current population into different kinds of fish (Note that the partition records numbers of individuals, not biomass). The fish are categorised by various characters. Traditionally these characters have been: length bins or age class, sex, maturity, area, stock, tag, and growth-path. However Casal2 has no predefined characters in the partition. This is a major extension from CASAL where users can extend on traditional problems, for example incorporating predator, prey, and in the case of some shellfish species a clock (death) category.

When defining the partition the user must choose:

- whether the partition records numbers by length class or age class (not both). This is the difference between an age based and length (size) based model
- which of the other characters are included in the partition, e.g., the number of areas, stocks, tagging events, or growth paths (if any of these characters are included in the partition).

The resulting partition can be conceptualised as a group of vectors, where each category is represented as a vector and the size of the vector is the number of ages or length classes, shown in Figure 4.1. Each element in a vector represents the number of individuals for that category in that age or length bin. splitting the partition up into separate categories allows categories to have different age or length structures.

The names and number of categories are user defined, but there must be at least one category defined for a model. The ages are defined as a sequence from age_{min} to age_{max}, with the last age optionally a plus group. In order to calculate biomass, the age-length relationship for each category must also be defined for an age based model. An example of how this is specified for four categories based on sex and area is as follows,

```
@categories
format mature.sex
names  spawn.male  spawn.female  nonspawn.male  nonspawn.female
age_lengths  male_AL female_AL   male_AL female_AL
```

For an example of these ideas, consider a model of a single fish population with a spawning and non-spawning fishery. The non-spawning fishery happens over most of the year (say 10 months) in the home area. The mature fish then migrate to the spawning area, where the spawning fishery operates. At the end of spawning, these fish, along with the recruits from the previous year, migrate back to the home area. The modeller decides that fish will be divided in the partition by age, sex, maturity, and area (spawning and home grounds). So the partition has 8 rows (2 sexes (mature or immature) 2 areas) and one column per age class.

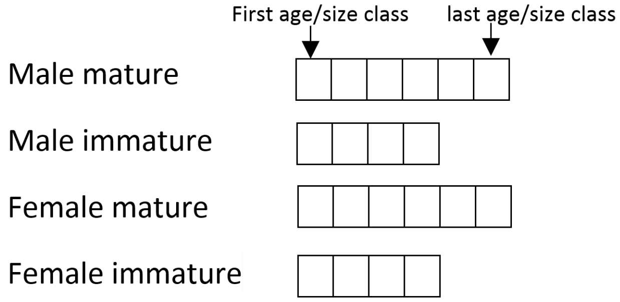

Figure 4.1: A visual representation of a partition

So they define four time steps, labelled 1 through 4. Step 1 includes the non-spawning fishery. Step 2 includes the migration to the spawning area. Step 3 includes the spawning fishery. Step 4 includes recruitment and the migration back to the home area. (In fact, they could have used only 3 time steps, by using a single step in place of their steps 2 and 3. Because the default order of processes within a time step places migrations before fisheries, the processes would still have occurred in the right order.) There are other details to be sorted out, such as the proportion of natural mortality occurring in each time step, but this gives the basic idea.

This structure can be used to implement complex models, with intermingling of separate stocks, with complex migration patterns over multiple areas, and multiple fisheries using different fishing methods and covering different areas and times. Note that there is little point in using a complex structure to model a stock when there are no observations to support that structure. In other words, use a structure for your model that is compatible with the data available.

The model is run from an initial year up to the final(current) year. It can also be run past the final year to make projections things that happen in the future up to the final projection year.

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

```
@model
start_year
final_year
min_age 1
max_age 20
age_plus_group True
initialisation_phases iphase
time_steps step1 step2
```

4.3. The state object and the partition

The key component of the state object is the partition, a group of vectors that store numbers of fish at age or length for a specific category. A category represents a group of fish that have specific attributes, examples of such attributes include life histories and growth paths. Characters in a population that display different attributes and that can make up a category or separate categories are:

- Sex (male or female);
- Area (any number of areas, named by the user);
- Stock (any number of stocks, named by the user);
- Maturity (immature or mature);
- Growth-path (any number of growth-paths);
- Tag (any number of tagging events);
- trophic level (Prey and Predator)

A stock is defined as a subpopulation of fish which recruits separately. See Section 4.11 for the treatment of maturity when it is not a character in the partition.

Growth-paths are a feature used to implement some persistence of length at age in an age-based model that uses some length/size data. Each growth-path has its own growth curve, and the length-based model features will consequently have different effects on different growth-paths. So, you need to tell Casal2 the following:

- Whether the model is age- or length-based.
- The number and nature of length classes in a length-based model.
- The minimum and maximum age classes in an age-based model.
- Whether there is a plus group.
- The names of all categories and there corresponding growth path labels.
- Whether the partition is divided by sex.
- Whether the partition is divided by maturity.
- Whether the partition has growth-paths, and, if so, how many.
- Whether the partition has multiple stocks, and, if so, how many, and their names.
- Whether the partition has multiple areas, and, if so, how many, and their names.
- Whether the partition includes tagged fish, and, if so, how many, and the names of the tag partitions.

Age classes are always 1 year wide, except that the maximum age group can optionally be a plus group. Users need to choose the minimum and maximum age classes. Length classes are defined by the user, and you need to specify how many length classes there are, the lower bound of each length class, and whether the last length class is a plus group, or if not, what its upper bound is. The relevant parameters are `class_mins` and `plus_group`. The `class_mins` parameter contains the lower bound of each class, and concludes with the upper bound of the last class if it is not a plus group. If, for example, length classes of 30-40, 40-50, 50-60, and 60+ cm were desired, you would set `class_mins 30 40 50 60` and `plus_group true`. Whereas if 30-40, 40-50, 50-60, and 60-70 cm were desired, you would set `class_mins 30 40 50 60 70` and `plus_group false`.

Casal2 allows categories of the partition to exist for certain years of the model. This is added for computational efficiency, when models contain a large number of categories that do not persist for all model years. Situations where this is beneficial is when a model contains a process that does a one off transition of fish from one category into another category, for example tagging events. Excluding categories for certain years saves initialising empty categories. This can be a big time saver if initialisation is run for 50 years and there are many tagging events.(When the ross sea model is up and running % differences would be a nice insert here)

Another important component of the state object in Casal2 are derived quantities. This includes quantities such as spawning stock biomasses (SSBs, mid-spawning season biomasses of spawning fish) for each or a combination of categories. Casal2 derives through the command `@derived_quantity`, this is needed if there is a stock-recruitment relationship.

4.4. Time sequences

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

- Initialisation
- Model run years
- Projection years

Annual cycle

The annual cycle is implemented as a set of processes that occur, in a user-defined order, within each year. Time-steps are used to break the annual cycle into separate components, and allow observations to be associated with different sets of processes. Any number of processes can occur within each time-step, in any order and can occur multiple times within each time-step. Note that time-steps are not implemented during the initialisation phases (effectively, there is only one time-step), and that the annual cycle in the initialisation phases can be different from that which is applied during the model years.

4.4.1. Initialisation

There are multiple methods to initialise a partition in Casal2. These methods are: iterative, fixed, derived and Cinitial. Model initialisation can occur in several phases, each of which can be a different type. At the end of the initialisation step, Casal2 runs through the model years carrying out processes in the order defined in the annual cycle, and can evaluate expected values of observations in order to calculate likelihoods, project forward to determine future states, or simulate observations from the current state.

Iterative Intitialisation

One of Casal2 methods for initialising the initial equilibrium state as an iterative process: a general solution that initialises complex structured models can be difficult to implement using analytic techniques. However, initialising via iteration for a long-lived species with complex transitions can take many iterations and be slow to run. In Casal2, 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 processes. Note that the length of the initialisation period may affect the model outputs, and that a period should be chosen to allow the population state to converge.

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

$$\widehat{\lambda} = \frac{\sum_i \sum_j |\text{element}(i,j)_t - \text{element}(i,j)_{t-1}|}{\sum_i \sum_j \text{element}(i,j)_t} \tag{4.1}$$

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) 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 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_partition`), 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 an iterative initialisation you need to define;

- The initialisation phases.
- The number of years in each phase and the processes to apply in each (default is the annual cycle).

Derived Intitialisation

`Derived` intitialisation is an analytical solution to calculate the equilibrium plus group using a geometric series. The benefit of this method is it can be solved in max_age - min_age +1 years, so is computationally faster than the iterative initialisation phase. Users should be warned that we have found under some process combinations (One way migrations). This solution does not reach the exact equilibrium partition. We advise to use this method for the computation benefits, but you should always compare to an iterative initialisation to satisfy the assumption that the partition is at an equilibrium state.

Cinitial Intitialisation

This phase can only be applied once a derived or iterative intialisation phases has been implemented. It works off an equilibrium state and uses Cinitial factors that can be estimated to shift the initial population away from an equilibrium state prior to start year. If there is known exploitation before data exists for a population this can be a solution for estimating a non equilibrium population. To apply this method, an observation of age composition data should be provided at start_year in order to estimate this non equilibrium population. This is implemented in the Southern blue whiting stock assessment for the Bounty Platform stock Dunn and Hanchet (2015).

Fixed Intitialisation

This is a user defined table that is taken to be the initial partition prior to start year. Users have the ability to initialise models by specify the numbers at age or length for each category. See initialisation type `state_category_by_age` for how to implement this initialisation phase.

4.5. Model run years

Following initialisation, the model then runs over a number of user-defined years from (initial_year to final_year). For this part of the model, the annual cycle can be broken into separate time-steps, and observations can be associated with the state of the model at the end of any time-step, i.e., likelihoods for particular observations are evaluated, if required, at the end of each time-step.

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

- The time-steps and the processes applied in each

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

4.6. Projection years

Projection years follow model run years. Projecting is the process of running the model forwards into the future, using stochastic and or deterministic values for population dynamic parameters, such as recruitments and catches. In a projection run in Casal2 a model is initialised and run through the model years from `initial` to the `final`. Then, the model is re run from `intial` to `projection_final_year`, where any parameter can be fixed or drawn from a stochastic process between this time period. Casal2 does not have default projections. Users must specify them using the `@project` blocks. This is important for parameters that are year specific such as year class parameters. If there is no `@project` for these parameters, they will not exist after `final_year` processes that call them will cause nonsensical output. Any estimable parameter can be projected forward. The types of processes where parameters are drawn during the projection years are: constant, LogNormal and empirical sampling.

Constant

A parameter can be fixed during all projection years or can be individually specified for each projection year. This is a deterministic process, where we are assuming the parameter is known without error for future years.

Empirical Sampling

Parameters that are of type vector or map are re sampled with replacement between a year range for projected years. This process redraws from an empirical distribution.

LogNormal

The randomised parameter are lognormally distributed, with mean 1, and specified standard deviation and autocorrelation on the log-scale. $\text{YCS}_i = exp(X_i)$, where (X_i) are generated as a Gaussian AR(1) process with standard deviation σ_R and mean - 0.5 σ_R (so that the mean of YCS_i is 1), and autocorrelation ρ. Set $\rho = 0$, the default, if you dont want autocorrelation. If the randomised parameter are modified by an arbitrary multiplier, then the only change is that parameter will have mean μ, where μ is the recruitment multiplier.

4.6.1. Single stepping Casal2

Casal2 has the ability to pause after each annual cycle and query the user to input new estimable parameters for the next annual cycle. This is an active area of development for Casal2 currently. The aim of this functionality is to be able to talk to another program such as **R**. **R** can read reports such as derived quantities after each annual cycle from Casal2 then using harvest control rules set catches for future years. For running management catch rules into the future.

4.7. Population processes

Population processes are those processes that change the population state of individuals. Processes produce changes in the model partition, by adding, removing or moving individuals between ages or categories. 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.

There are two types of processes, processes that occur across multiple time steps in the annual cycle e.g Natural Mortality and Instantaneous Mortality. There are also processes that only occur within the time step they are defined. 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.

4.7.1. Recruitment

Recruitment processes are defined as a process that introduces new individuals into the model. Casal2 implements two types of recruitment process, constant recruitment and Beverton-Holt recruitment (Beverton and Holt, 1957).

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

For the constant and Beverton-Holt recruitment processes, the number of individuals following recruitment in year y is,

$$N_{i,j} \leftarrow N_{i,j} + p_j(R_y) \tag{4.2}$$

where $N_{i,j}$ is the numbers in category j at age i, p_j is the proportion to category j, and R_y is the number of recruits for year y. See below for how R_y is determined in each of these cases.

Constant Recruitment

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

$$R_y = R_0 \tag{4.3}$$

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

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

```
@process Recruitment
type constant_recruitment
categories immature
proportions 1.0
r0 500000
age 1
```

Beverton-Holt recruitment

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

$$R_y = R_0 \times YCS_{y-\text{ssb\_offset}} \times SR(SSB_{y-\text{ssb\_offset}}) \tag{4.4}$$

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

$$SR(SSB_y) = \frac{SSB_y}{B_0} / \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB_y}{B_0}\right)\right) \tag{4.5}$$

Note that the Beverton-Holt recruitment process requires a value for B_0 and SSB_y to resolve the stock-recruitment relationship. Here, a derived quantity (see Section 4.8) must be defined that provides the annual SSB_y for the recruitment process. B_0 is then defined as the value of the SSB at the end of one of the initialisation phases. During initialisation the YCS multipliers are assumed to be equal to one, and recruitment that happens in the initialisation phases that occur before and during the phase when B_0 is determined is assumed to have steepness $h = 1$ (i.e. in those initialisation phases, recruitment is simply equal to R_0). Recruitment in the initialisation phases after the phase where B_0 was determined follow the Beverton-Holt stock-recruit relationship defined above. R_0 and B_0 have a direct relationship when there are no density dependent processes, for this reason users can choose to initialise models using B_0 or $R_0\dot{I}$n New Zealand B_0 is often used, as biological reference points for managing marine populations is based on a percentage of B_0.

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

$$YCS_i = \begin{cases} Y_i / mean_{y \in S} & : y \in S \\ Y_i & : y \notin S \end{cases}$$

Note that the an effect of this parameterisation is that R0 is then defined as the mean estimated recruitment over the years S, because the mean year class multiplier over these years will always be one.

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

```
@process Recruitment
type recruitment_beverton_holt
categories immature
proportions 1.0
r0 500000
b0_intialisation_phase phase1
steepness 0.75
age 1
ssb SSB_Biomass
standardise_ycs_years 1994-2004
ycs_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
YCS_values   1    1    1    1    1    1    1    1    1    1    1    1    1
```

- If recruitment then ageing then spawning, then `ssb_offset` should equal `min_age` + 1.
- If spawning then ageing then recruitment, then `ssb_offset` should equal `min_age` - 1.
- If any other order is used, then `ssb_offset` should equal `min_age`.

If you have more than one ageing process and a bevertonholt recruitment process you will be warned to set your own `ssb_offset` as Casal2 will set `ssb_offset` based upon the first ageing process which may be not want the user desires.

4.7.2. Ageing

The ageing process simply moves all individuals in the named categories i to the next age class $j+1$. The ageing process is defined as,

$$\text{element}(i,j) \leftarrow \text{element}(i,j-1) \tag{4.6}$$

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

$$\text{element}(i,age_{\max}) \leftarrow \text{element}(i,age_{\max}) + \text{element}(i,age_{\max-1}). \tag{4.7}$$

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 Casal2 by default. As with other processes, Casal2 will not apply a process unless its defined and specified as a process within the annual cycle. Hence, it is possible to specify a model where a category is not aged. Casal2 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 Casal2, constant rate, event, biomass-event and instantaneous. These processes remove individuals from the partition, either as a rate, as a total number (abundance), as a biomass of individuals or as a mixture of these. Casal2 does not implement the Baranov catch equation yet. To apply both natural and biomass-event mortality, users can use `mortality_instantaneous`.

All mortality processes occur within the mortality block of a time step. A mortality block is a number of consecutive mortality processes in a single time step. If no mortality processes occur in a time step then this block defaults to the end of the time step. Casal2 will error out if you have multiple mortality processes in a time step that are not consecutive. This mortality block is important for derived quantities see Section 4.8.

4.7.3.1. Constant mortality rate

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

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

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

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

4.7.3.2. Event and biomass-event mortality

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

Casal2 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 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 i, with the numbers removed at age j determined by a selectivity-at-age S_j is applied as follows:

First, calculate the vulnerable abundance for each category i in $1 \ldots I$ for ages $j = 1 \ldots J$ that are subject to event mortality,

$$V(i,j) = S(j)N(i,j) \tag{4.8}$$

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

$$V_{total} = \sum_i \sum_j V(i,j) \tag{4.9}$$

Hence the exploitation rate to apply is

$$U = \begin{cases} C/V_{total}, & \text{if } C/V_{total} \le U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.10}$$

And the number removed R from each age j in category i is,

$$R(i,j) = UV(i,j) \tag{4.11}$$

For example, to specify fishing mortality based on catches given for each year, over categories ‘immature’ and ’mature’, with selectivity ‘FishingSel’ and assuming a maximum possible exploitation rate of 0.7, then the syntax is

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

4.7.3.3. Instantaneous mortality

The instantaneous mortality process is a combination of natural mortality and event biomass mortality that occurs across multiple time steps. This process applies half the natural mortality, then to apply the mortalities from all the fisheries instantaneously, then to apply the remaining half of the natural mortality.

When Instantaneous mortality is applied the following equations are used.

- An exploitation rate (actually a proportion) is calculated for each fishery, as the catch over the selected-and-retained biomass,

$$U_f = \frac{C_f}{\sum_j \bar{w}_j S_{f,j} n_j e^{-0.5tM_j}}$$

- The fishing pressure associated with fishery f is defined as the maximum proportion of fish taken from any element of the partition in the area affected by fishery f,

$$U_{f,obs} = max_j(\sum_k S_{k,j} U_k)$$

  where the maximum is over all partition elements affected by fishery f, and the summation is over all fisheries k which affect the jth partition element in the same time step as fishery f.

  In most cases the fishing pressure will be equal to the exploitation rate (i.e., $U_{f,obs} = U_f$). This will not be true only if (a) there is another fishery operating in the same time step as fishery f and affecting some of the same partition elements, and/or (b) the selectivity $S_{f,j}$ does not have a maximum value of 1.

  There is a maximum fishing pressure limit of Uf,max for each fishery f. So, no more than proportion Uf,max can be taken from any element of the partition affected by fishery f in that time step. Clearly $0 \leq U_{max} \leq 1$. It is an error if two fisheries which affect the same partition elements in the same time step do not have the same Umax.

  For each f, if $U_{f,obs} > U_{f,max}$, then U_f is multiplied by $U_{f,max}/U_{f,obs}$ and the fishing pressures are recalculated. In this case the catch actually taken from the population in the model will differ from the specified catch, C_f.

- The partition is updated using

$$n'_j = n_j exp(-tM_j)\left[1 - \sum_f S_{f,j} U_f\right]$$

An example of the syntax is if we want to apply natural mortality of 0.19 across three time steps on both male and female categories. And we have two fisheries `FishingWest FishingEast` with there respective catches known for years 1975:1977 in kilograms. These are given in the `catches` table and information on selectivities, penalties and maximum exploitation rates are given in the `fisheries` table.

```
@process instant_mort
type mortality_instantaneous
m 0.19
time_step_ratio 0.42 0.25 0.33
selectivities One
categories male female
units kgs


table catches
year FishingWest FishingEast
1975 80000 111000
1976 152000 336000
1977 74000 1214000
end table


table fisheries
fishery       category  selectivity u_max   time_step penalty
FishingWest   stock     westFSel    0.7     step1     CatchPenalty
FishingEast   stock     eastFSel    0.7     step1     CatchPenalty
end_table
```

4.7.3.4. Baranov Mortality

Coming Soon!

4.7.4. Transition By Category

This process covers two major processes being, Maturation and Migration. Because Casal2partition is up to the user this type of process should only be used if maturity and/or area are defined in the partition. This processes moves individuals from one category to another, for the case of maturity, this could be moving individuals form immature category to mature category. For migration this could be moving individuals from an area defined category to another.

4.7.4.1. Maturation

Maturation is the process in which immature fish become mature and are moved accordingly in the partition. See Section refsec:maturity-notinpartition for how to treat maturity when it is not a character in the partition.

```
@process Maturation
type category_transition
from male.immature
to male.mature
selectivities MatureSel matureSel
proportions 1 1
```

4.7.4.2. Migration

Migration is the process of moving fish from one area to another. For this to be sensibly applied in Casal2 there needs to be a category for the source area and a category for the target area. If two or more migrations are specified in the same time step, then they take place in the order in which they are given.

Aslong as there is one to one category relationship. That is for every source category there is one target category. You can state that a given proportion of these fish migrate (constant across all age or length classes), or provide a selectivity of proportions migrating by age or length class.

$$N_{a,j} = N_{a,i} \times P_i \times S_{a,i} \tag{4.12}$$

where $N_{a,j}$ is the number of individuals that have moved to category j from category i in age a and $N_{a,i}$ is the number of individuals in category i. P_i is the proportion parameter for category i and $S_{a,i}$ is the selectivity at age a for category i.

An example, to specify a simple spawning migration of mature males from a western area migrating to an eastern (spawning) area, then the syntax is

```
@process Spawning_migration
type category_transition
from West.males
to East.males
selectivities MatureSel
proportions 1
```

Where `MatureSel` is a selectivity that describes the proportion of age or length classes that are mature and thus move to the eastern area.

4.7.5. Tag Release events

Tagging processes can be age or length based processes, where by numbers of fished are moved from an untagged category to a tagged category that the user has defined in the `@Categories` block. Tag release processes can also account for tag induced mortality on individuals. Age based tag release events take a known number of individuals tagged for each age and do a straightforward category transition along with extra mortality. Length based tag release processes are more complicated, as Casal2 needs to calculate the age length matrix and exploitation by each length to then move the correct numbers at age based on a length input.

4.7.6. Tag Loss

Tag Loss is the process where tags are lost from tagged categories over time from tag failure or getting knocked off. This process is applied as a instantaneous mortality rate that can happen over multiple time steps in the annual cycle. This method assumes when tags are lost that the fish is removed from the partition. All though this seems logically incorrect, we are dealing with such a small number of fish that the impact is minimal and computationally simpler. Note that if your tagging events make up a large proportion of the population you may want to adjust this method. There will be two types of tag loss processes that are termed `single` and `double`. Currently only `single` exists in Casal2. `double` will deal with situations where a tag release process tags individuals with two tags. In which there is another formulae to work out the rate of tag loss.

```
@process Tag_loss
type tag_loss
```

```
categories tagged_fish
tag_loss_rate 0.02
time_step_ratio 0.25 0.75
selectivities One
tag_loss_type single
year 1985
```

4.8. Derived quantities

Some processes require, as arguments, a population value derived from the population state. These are termed `derived quantities`. Derived quantities are values, calculated by Casal2 as the end of a specified time-step in every year, and hence they have a single value for each year of the model. Derived quantities can be calculated as either an abundance or as a biomass. Abundance derived quantities are simply the count or sum of categories (after applying a selectivity). Biomass derived quantities are similar, except they are a measure of biomass. Derived quantities are also calculated during the initialisation phases, and hence the time-step during each phase must also be specified. If the initialisation time-steps are not specified, Casal2 will calculate the derived quantity during the initialisation phases in every year, at the end of the annual cycle.

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

Derived quantities are associated with a mortality block see section 4.7.3 for more detail on mortality blocks. Users can ask for derived quantities partway through mortality blocks. Currently two methods are implemented in Casal2 to interpolate derived quantities part-way through a mortality block, these are `weighted_sum` and `weighted_product`, they are defined as,

- `weighted_sum`: after proportion p of the mortality block, the partition elements are given by $n_{p,j} = (1-p)n_j + p'_j$
- `weighted_product`: after proportion p of the mortality block, the partition elements are given by $n_{p,j} = n_j^{1-p} n'^{p}_j$

where, n_p, j is the derived quantity at proportion p of the mortality block for category j. n_j is the quantity at the beginning of the mortality block and n'_j is the quantity at the end of the mortality block.

As an example, to define a biomass derived quantity (say spawning stock biomass, SSB) for a model, evaluated at the end of the first time-step (labelled `step_one`), over all 'mature' male and female categories and halfway through the mortality block using the `weighted_sum` method, we would use the syntax,

```
@derived_quantity SSB
type biomass
time_step step_one
categories mature.male mature.female
selectivities One
time_step_proportion 0.5
time_step_proportion_method weighted_sum
```

4.9. Age-length relationship

The age-length relationship defines the length at age (and the weight at length, see Section 4.9) of individuals at age/category within the model. There are three length-age relationships available in Casal2. The first is the naive no relationship (where each individual has length 1 irrespective of age). The second and third are the von-Bertalanffy and Schnute relationships respectively. The length-at-age relationship is used to determine the length frequency, given age, and then with the length-weight relationship, a weight-at-age of individuals within an age/category.

The three age-length relationships are,

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

von Bertalanffy: where length at age is defined as,

$$\bar{s}(age) = L_\infty \left(1 - \exp\left(-k\left(age - t_0\right)\right)\right) \tag{4.13}$$

Schnute: where length at age is defined as,

$$\bar{s}(age) = \begin{cases} \left[y_1^b + (y_2^b - y_1^b)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right]^{1/b}, & \text{if } a \neq 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right], & \text{if } a \neq 0 \text{ and } b = 0 \\ \left[y_1^b + \left(y_2^b - y_1^b\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right]^{1/b}, & \text{if } a = 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right], & \text{if } a = 0 \text{ and } b = 0 \end{cases} \tag{4.14}$$

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 lengths at reference ages τ_1 and τ_2, and a and b (when $b = 1$, this reduces to the von Bertalanffy with $k = a$).

When defining length-at-age in Casal2, you must also define a length-weight relationship (see Section 4.9 below).

Calculation of length-at-age (in an age-based model)

Interpolation of length-at-age

Size-weight relationship

There are two length-weight relationship,s available in Casal2. The first is the naive no relationship. Here, the weight of an individual, regardless of length, is always 1. The second is the basic relationship.

The two length-weight relationships are,

- None: The length-weight relationship where

$$\text{mean weight} = 1 \tag{4.15}$$

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

  $$w = al^b \tag{4.16}$$

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

  $$w = (al^b)(1 + cv^2)^{\frac{b(b-1)}{2}} \tag{4.17}$$

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

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

Calculation of mean weight

4.10. Weightless model

4.11. Maturity, in models without maturing in the partition

If maturity is not a character of the partition it can easily be derived at in instance in time using selectivities. Applying a maturity selectivity on to the partition allows Casal2 to use mature elements in processes, derive mature biomasses estimates (using derived quantities), and report the mature partition as an output.

4.12. Selectivities

A selectivity is a function that can have a different value for each age class. Selectivities are used throughout Casal2 to interpret observations (Section 5) or to modify the effects of processes on each age class (Section 4). Casal2 implements a number of different parametric forms, including logistic, knife edge, and double normal selectivities. Selectivities are defined in there own command block (`@selectivity`), where the unique label is used by observations or processes to identify which selectivity to apply.

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 selectivities can be length based, However Caution, more testing is needed for this functionality.

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). Casal2 implements range checks on some parameters to test for a possible numeric overflow error before attempting to calculate function values. For example, the logistic selectivity is implemented such that if $(a50 - x)/ato_95 > 5$ then the value of the selectivity at $x = 0$, i.e., for $a50 = 5$, $ato_95 = 0.1$, then the value of the selectivity at $x = 1$, without range checking would be 7.1×10^{-52}. With range checking, that value is 0 (as $(a50x)/ato_95 = 40 > 5$).

The available selectivities are;

- Constant
- Knife-edge
- All values
- All values bounded
- Increasing
- Logistic
- Inverse logistic
- Logistic producing
- Double normal
- Double exponential
- Cubic spline (Not yet implemented)

The available selectivities are described below.

4.12.1. `constant`

$$f(x) = C \tag{4.18}$$

The constant selectivity has the estimable parameter C.

4.12.2. `knife_edge`

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{4.19}$$

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`

$$f(x) = V_x \tag{4.20}$$

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`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \tag{4.21}$$

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`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \leq x \leq H \\ f(\alpha), & \text{if } x \geq H \end{cases} \tag{4.22}$$

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`

$$f(x) = \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.23}$$

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. `inverse_logistic`

$$f(x) = \alpha - \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.24}$$

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

4.12.8. `logistic_producing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1)) / (1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \tag{4.25}$$

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

4.12.9. `double_normal`

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

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 $x = a_1 + s_R$.

4.12.10. `double_exponential`

$$f(x) = \begin{cases} \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \leq x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{4.27}$$

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

4.12.11. `spline`

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

4.13. Time Varying Parameters

Casal2 has the functionality to vary a parameter annually between the start and final year of a model run. This can be for blocks of years or specific years if chosen. For years that are not specified the parameter will default to the input or if in a iterative state such as estimation mode, the value being trialled at that iteration. Available methods for time varying a parameter. Where this functionality will become quite useful is in simulating more realistic observations. When you allow fisheries to have annual varying catchabilities and other more realistic model components simulated observations become more real data and thus conclusions based on simulated data are more useful.

4.13.1. `Constant`

Allows a parameter to have an alternative values during certain years, which can be estimated.

4.13.2. `Random Walk`

A random deviate added into the last value drawn from a standard normal distribution. This has an estimable parameter σ. For reproducible modelling, it is highly recommended that users set the seed (see Section 3.4) when using stochastic functionality like this, otherwise reproducing models becomes very difficult.

4.13.3. `Exogenous`

parameters are shifted based on an exogenous variable, an example of this is fishing selectivity parameters that may vary between years based on known changed fishing behaviours such as fishing season start time.

$$\delta_y = a(E_y - \bar{E}) \tag{4.28}$$

where δ_y is the shift in parameter X in year y, a is an estimable shift parameter, E is the exogenous variable and E_y is the value of this variable in year y. For more information readers can see Francis et al. (2003).

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 Casal2:

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

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

5.2. The objective function

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

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

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

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

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

5.3. Specifying the parameters to be estimated

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

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

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

See Section 3.5.5 for instructions on how to generate the parameter name. At least one parameter is to be estimated if doing an estimation, profile, or MCMC run. Initial values for the parameters to be estimated will still need to be provided, and these are used as the starting values for the minimiser. However, these may be overwritten if you provide a set of alternative starting values (i.e., using `casal2 -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 (`type`) (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. Bounds should be carefully chosen as they effect the space in which the minimisers search over. Minimisers convert lower and upper bound into a minimisation space (for example -1,1 space). If estimating only some elements of a vector, either define each element of the vector to be estimated (see 3.5.5) or fix the others by setting the bounds equal.

5.4. Point estimation

Point estimation is invoked with `casal2 -e`. Mathematically, it is an attempt to find a minimum of the objective function. Casal2 has multiple algorithms for solving (minimising) the optimisation problem. There are three non auto differential minimisers: numerical differences, differential evolution minimiser, and the dlib minimiser. There are also three auto differential minimisers being: ADOL-C, CPPAD, and BETADIFF. For references see section 1.7

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 `casal2 -i`.

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

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

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

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

5.4.2. The differential evolution minimiser

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

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

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

5.4.3. Betadiff minimiser

An auto-differentiable minimiser for non-linear models, This is the minimiser from the original CASAL package.

5.4.4. ADOL-C minimiser

An auto-differentiable minimiser for non-linear models.

5.4.5. CPPAD minimiser

An auto-differentiable minimiser for non-linear models.

5.4.6. Dlib minimiser

Non auto-diff minimiser

5.5. Posterior profiles

If profiles are requested `casal2 -p`, Casal2 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, Casal2 will fix its value at a sequence of n evenly spaced numbers ($step$) between a specified lower and upper bounds l and u, and calculate a point estimate at each value.

By default $step = 10$, and $(l,u) =$ (lower bound on parameter plus $(range/(2n))$, upper bound on parameter less $(range/(2n))$. Each minimisation starts at the final parameter values from the previous resulting value of the parameter being profiled. Casal2 will report the objective function for each parameter value. Note that an initial point estimate should be compared with the profile, not least to check that none of the other points along the profile have a better objective function value than the initial 'minimum'.

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

You can also supply the initial starting point for the estimation using `casal2 -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 Casal2 and compare the results.

5.6. Bayesian estimation

check and confirm text
KL comment: This text is from SPM and is nearly verbatim S6.5 in CASASL, but two large sections exluded: ...request covariate matrix change adaptively... and from ...multivariate t dist... onwards. OK to use/ Additions required?

Casal2 can use a Monte Carlo Markov Chain (MCMC) to generate a sample from the posterior distribution of the estimated parameters `casal2 -m` and output the sampled values to a file (optionally keeping only every nth set of values).

As Casal2 has no post-processing capabilities. Casal2 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 Casal2. See Section 9.3 for a better description of the sequence of Casal2 commands used in a full Bayesian analysis.

Casal2 uses a straightforward implementation of the Metropolis-Hastings algorithm (Gelman et al., 1995, Gilks et al., 1994). The Metropolis-Hastings algorithm attempts to draw a sample from a Bayesian posterior distribution, and calculates the posterior density π, scaled by an unknown constant. The algorithm generates a 'chain' or sequence of values. Typically the beginning of the chain is discarded and every Nth element of the remainder is taken as the posterior sample. The chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

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

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

The user can specify the starting point of the point estimate minimiser using `casal2 -i`. Don't start it too close to the actual estimate (either by using `casal2 -i`, or by changing the initial parameter values in input configuration file) as it takes a few iterations to form a reasonable approximation to the Hessian.

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

- Start from the point estimate.
- Start from a random point near the point estimate (the point is generated from a multivariate normal distribution, centred on the point estimate, with covariance equal to the inverse Hessian times a user-specified constant). This may be useful if the chain gets 'stuck' at the point estimate, or if you wish to generate multiple chains from for later MCMC diagnostic tests.
- Start from a point specified by the user with `casal2 -i` (*was NYI, to be included?*)

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

- The covariance matrix is taken as the inverse of the approximate Hessian from the quasi-Newton minimiser.
- The covariance matrix is modified so as to decrease all correlations greater than `@mcmc.max_correlation` down to `@mcmc.max_correlation`, and similarly to increase all correlations less than -`@mcmc.max_correlation` up to -`@mcmc.max_correlation` (the `@mcmc.max_correlation` parameter defaults to 0.8). This should help to avoid getting 'stuck' in a lower-dimensional subspace.
- The covariance matrix is then modified either by,
  - if `@mcmc.adjustment_method=covariance`: that if the variance of the ith parameter is non-zero and less than `@mcmc.min_difference` times the difference between the

parameters' lower and upper bound, then the variance is changed, without changing the associated correlations, to $k = \text{min\_diff}(upper\_bound_i - lower\_bound_i)$. This is done by setting

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

for $i \neq j$, and $\text{var}(i)' = k$

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

This allows each estimated parameter to move in the MCMC even if its variance is very small according to the inverse Hessian. In both cases, the `@mcmc.min_difference` parameter defaults to 0.0001.

- The `@mcmc.stepsize` (a scalar factor applied to the covariance matrix to improve the acceptance probability) is chosen by the user. The default is $2.4d^{-0.5}$ where d is the number of estimated parameters, as recommended by Gelman et al. (Gelman et al., 1995). However, you may find that a smaller value may often be better.

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

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

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 Casal2 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). (*was NYI, to be included?*)

The variance-covariance matrix of this sub-sample of chain is calculated. As above, correlations greater than `@mcmc.max_correlation` are reduced to `@mcmc.max_correlation`, correlations less than `@mcmc.max_correlation` are increased to `@mcmc.max_correlation`, and very small non-zero variances are increased (`@mcmc.covariance_adjustment` and `@mcmc.min_difference`. The result is the new variance-covariance matrix of the proposal distribution. (*was NYI, to be included?*)

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

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. (*was NYI, to be included?*)

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

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

If you specify the start of the chain using `casal2 -i`, these fixed parameters are set to the values in the file. (*was NYI, to be included?*)

The posterior sample can be used for (projections (Section 4.6)(*was NYI, to be included?*)) or simulations (Section 6.7) with the values supplied using `casal2 -i` *`file`*.

(*following from CASAL, to be included?*)

```
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.

Having produced one or more Markov chains and looked at the diagnostics, reload all the chain
    output files into CASAL and use them to generate a single posterior sample (using -C). At
    this stage, the first burn_in iterations for each chain are discarded (so, with keep 10,
    burn_in 1000, the first 1000 recorded samples are discarded for each chain). Unless a very
     large value of keep was originally chosen, it will be necessary to further reduce the
    size of the posterior sample (possibly down to several hundred) such that it can be
    analysed in a reasonable amount of time. This is done by sub-sampling. You specify the
    size of the sub-sample to be produced (or else no sub-sampling is done). You have the
    option to generate a systematic sub-sample (i.e., every nth point is kept) or a random sub
    -sample (the former is recommended except with prior re-weighting, when the latter must be
     used).

Given a posterior (sub)sample, CASAL can calculate a list of output quantities for each sample
     point (see Section 7.2). These quantities can be dumped into a file (using casal -v) and
    read into an external software package where the posterior distributions can be plotted
    and/or summarised.

The posterior sample can also be used for projections (Section 7.3) and stochastic yield
    calculations (Section 7.5). The advantage of this is that the parameter uncertainty, as
    expressed in your posterior distribution, can be included into the risk and yield
    estimates.

It is possible to investigate the results that would have been obtained if a different prior
    had been specified This is called prior re-weighting and is done by calculating the ratio
    of the new prior to the original prior for each point in the posterior sample, then using
    these ratios as probability weights when generating a random (not systematic) sub-sample
    with casal -C. Prior re-weighting is applicable only if the new prior is zero in every
    part of the parameter space for which the original prior was zero. Also, it is likely to
```

```
be numerically unstable unless the new prior is very small in every part of the parameter
space for which the original prior was very small.
```

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.

Casal2 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 $s = \sqrt{\log(1+c^2)}$.

   (following from CASAL, to be included?)

   ```
   6. Normal-log with log(p) having mean m and standard deviation s,
   ```

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 $\tau \leq 0$.

(following from CASAL, to be included?)

```
Vectors of parameters can be independently (but not necessarily identically) distributed
    according to any of the above forms, in which case the joint negative-log-prior for the
    vector is the sum of the negative-log-priors of the components. Values of each parameter
    need to be specified for each element of the vector.

In addition, for a vector p of n identically distributed parameters (for example, YCS) the
    following priors are allowed:

1. Multivariate normal from a stationary AR(1) process with parameters
.
.
.

2. Multivariate normal-log, where log(p) forms a stationary AR(1) process as per 1. above,
    with parameters
.
.
.
3. Multivariate normal-log with mean 1, where E(pi)=1 and log(p) forms a stationary AR(1)
    process as for the multivariate normal above, with parameters
.
.
.
```

5.8. Penalties

Penalties are associated with processes and 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. Casal2 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`, `@process[label].type=tag_by_length` and `@process[label].type=category_transition` are the only penalties implemented.

For 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.

5.9. Additional Priors

Additional priors are the inverse

5.10. Estimate Transformations

Casal2 has the untested functionality of transforming an estimated parameter in a new space. This may be done to remove correlation for other convergence or optimisation purposes. This functionality transforms the estimate and the bounds to the transformed space along with the prior. To account for the change variable a Jacobian is added to the objective function. For more information uses are asked to read the STAN manual **REFERENCE**. The user must supply the type, bounds for the transformed varaiable can be supplied by the user, but if not Casal2 will work them out. NOTE must be used with caution. May be buggy!!!

5.10.1. `log`

5.10.2. `Inverse`

5.10.3. `Log odds`

5.10.4. `Simplex`

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.

Casal2 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 Casal2 calculates the expected values, and the likelihoods that are available for each type of observation.

Casal2 evaluates the observations at the end of a time-step (i.e., after all of the processes for that time-step have been applied). However, the observation can be applied to the abundance at the start of a time-step or part-way through a time-step by the use of the `proportion_time_step` subcommand.

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

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

6.2. Proportions-at-age observations

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

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

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

Proportions-at-age observations can be supplied as;

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

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

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

Proportions-at-age for a single category

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

```
categories male
```

Casal2 then expects that there will be a single vector of proportions supplied, with one proportion for each age class within the defined age range, and that these proportions sum to one.

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

The observations must be also supplied using all or some of the the values of defined by some *categorical* layer. Casal2 calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

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

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

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

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

Or, for both A and B as,

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

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

Proportions-at-age for multiple categories

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

```
categories male female
```

Casal2 then expects that there will be a single vector of proportions supplied, with one proportion for each category and age class combination, and that these proportions sum to one.

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

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

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

Proportions-at-age across aggregated categories

This form of the observation extends the idea above, but allows categories to be aggregated before the proportions are calculated. It is used to model a set of proportions from several categories that have been combined by age class. To indicate that two (or more) categories are to be aggregated, separate them with a '+' symbol. For example, to specify that the observations are of the proportions of male and females combined within each age class, then the subcommand `categories` for the `@observation[label].type=proportion_by_age` command is,

```
categories male + female
```

Casal2 then expects that there will be a single vector of proportions supplied, with one proportion for each age class, and that these proportions sum to one.

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

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

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

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

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

6.2.1. Likelihoods for proportions-at-age observations

Casal2 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

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

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 observations

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

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

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

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

```
categories male
```

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

defines that the proportion of males in each age class as a proportion of males + females. Casal2 then expects that there will be a vector of proportions supplied, with one proportion for each age class within the defined age range, i.e., if the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the the ages 3, 4, 5, 6, 7, 8, 9, and 10). The expected values will be the expected proportions of male to male + female within each of these age classes, after applying the selectivities at the year and time-step specified.

The observations must be supplied using all or some of the values defined by a categorical layer. Casal2 calculates the expected values by summing over the ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

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

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

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

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

Or, for both A and B as,

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

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

6.3.1. Likelihoods for proportions-by-category observations

Casal2 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;

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

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 or biomass observations

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

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

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

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

defines an observation for males after applying the selectivity male-selectivity. Casal2 then expects that there will be a single observation supplied. The expected values for the observations will be the expected abundance (or biomass) of males, after applying the selectivities, at the year and time-step specified.

The observations must be supplied using all or some of the the values of defined by a categorical layer. Casal2 calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories for those spatial cells where the categorical layer has the same value as defined for each vector of observations.

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

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

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

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

Or, for both A and B as,

```
@observation MyAbundance
type abundance
layer Area
...
categories male
obs A 1000
obs B 1200
...
```

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

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

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

6.4.1. Likelihoods for abundance observations

The lognormal likelihood

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

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

where

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

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

Casal2 can apply ageing error age frequency observations. Ageing error is applied to the expected values for proportions-at-age observations. The ageing error is applied as a misclassification matrix, which has the effect of 'smearing' the age frequencies. These are used in calculating the fits to the observed values, and hence the contribution to the total objective function.

Ageing error is optional, and if it is used, it may be omitted for any individual time series. Different ageing error models may be applied for different observation commands. See Section 7.11 for reporting the misclassification matrix.

The ageing error models implemented are,

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

Note that the expected values (fits) reported by Casal2 for observations with ageing error will have had the ageing error applied.

6.7. Simulating observations

Casal2 can generate simulated observations for a given model with given parameter values (using `spm -s`). Simulated observations are randomly distributed values, generated according to the error assumptions defined for each observation, around fits calculated from one or more sets of the 'true' parameter values. Simulating from a set of parameters can be used to generate observations from an operating model or as a form of parametric bootstrap.

The procedure Casal2 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 Casal2 will report simulated observations using the usual observation report (`@report[label].type=observation`). The report `@report[label].type=simulated_observation` will generate simulated observations in a form suitable for use as input within a Casal2 input configuration file. See Section 7 for more detail.

6.8. Pseudo-observations

Casal2 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 Casal2 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. Casal2 can also generate simulated observations from pseudo-observations. Note that;

- Output will only be generated if a report command `@report[label].type=observation` is specified.
- The observed values should be supplied (even if they are 'dummy' observation). These will be processed by Casal2 as if they were actual observation values, and must conform to the validations carried out for the other types of likelihood.
- The subcommands `likelihood`, `obs`, `error_value` and `process_error` have no effect when generating the expected values for the pseudo-observation.
- When simulating observations, Casal2 needs the subcommand `simulation_likelihood` to tell it what sort of likelihood to use. In this case, the `obs`, `error_value` and `process_error` are used to determine the appropriate terms to use for the likelihood when simulating.

7. The report section

The report section specifies the printouts and other outputs from the model. Casal2 does not, in general, produce any output unless requested by a valid `@report` block.

Reports from Casal2 can be defined to print partition and states objects at a particular point in time, observation summaries, estimated parameters and objective function values. See below for a more extensive list, and an example of an observation report.

```
@report observation_age ## label of report
type observation ## Type of report
observation age_1990 ## label corresponding to an @observation report, shown below

@observation age_1990
type proportion_at_age
year 1990
plus_group
etc ...
```

Reports from Casal2 all conform to a standard style (with one exception — the `output_parameters` report, see below). The standard style is that reports are prefixed with an asterix followed by a user-defined label and type of report in brackets (e.g., `*label (type)`), with the report ending with the line `*end`. For example,

```
*My_report(type)
...
*end
```

This syntax should make it easier for external packages to be configured to read Casal2 output. The `extract` functions in the **R** `CASAL2` package uses this information to identify and read Casal2 output.

Note that the `output_parameters` report does not print either a header or `*end` at the end of the report. This is as the `output_parameters` 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 Casal2 (with the command `casal2 -i`). This is a specialised report for `casal2 -o` command. For estimate values in standard output users are recommended to use `type=estiamte_value`.

Note that reports can be defined that may not be generated. For example printing the partition for a year and/or time-step that does not exist or reporting the covariance matrix when not estimating. Such reports are ignored by Casal2 and the program will not generate any output for these reports — although they must still conform to Casal2s syntax requirements.

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

7.1. Print the partition

Print the partition for a given year or given years and time-step. This prints out, the numbers of individuals in each age class and category in the partition for each year. Note that this report is evaluated at the end of the time-step in the given year(s).

7.2. Print the partition at the end of an initialisation

Print the partition following an initialisation phase. This prints out, the numbers of individuals in each age class and category in the partition following an initialisation phase.

7.3. Print a process summary

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

7.4. Print derived quantities

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

7.5. Print the estimated parameters

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

7.6. Print the estimated parameters in a vector format

Print the estimated parameter values out as a vector. The `estimate_values` report prints the name of the parameter, followed by the value of that run.

7.7. Print the objective function

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

7.8. Print the covariance matrix

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

7.9. Print observations, fits, and residuals

Prints out for each category or combination of categories, expected values as calculated by the model, residuals (observed − expected), the error value, process error, and the total error (i.e., the error value as modified by any additional process error), and the contribution to the total objective function of that individual point in the observation.

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

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

7.10. Print simulated observations

Prints out a complete observation definition (i.e., in the form defined by `@report[label].type=observation`), but with observed values replaced by randomly generated simulated values. The output is in a form suitable for use within a Casal2 input configuration file, reproducing the command and subcommands from the input configuration file.

7.11. Print the ageing error misclassification matrix

Prints out the ageing error misclassification matrix.

7.12. Print selectivities

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

7.13. Print the random number seed

Prints the random number seed used by Casal2 to generate the random number sequence. Future runs made with the same random number seed and the same model will produce identical outputs.

7.14. Print the results of an MCMC

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

7.15. Print the MCMC samples as they are calculated

Print the MCMC samples for each new *i*th sample as they are calculated while doing an MCMC. The output file will be updated with each new sample as it is calculated by Casal2.

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

Print the MCMC objective function values (along with the proposal covariance matrix) for each new *i*th sample as they are calculated while doing an MCMC. The output file will be updated with each new set of objective function values as it is calculated by Casal2.

7.17. Tabular reporting

An alternative reporting framework to the standard output is the tabular reporting. Tabular reporting is used with multiline `-i` input files (like the MCMC reports). Tabular reports will print out a row that will correspond with each row of the `-i` input files. Tabular reporting is is in invoked at the command line using the following command `casal2 -r --tabular -i file_name`. Currently derived quantities and estimate_values are the only report types that are within this framework. For each input file the output will begin with the names of each column followed by a multiline report ending with the *end syntax. These tables can be easily read into **R** using the `CASAL2` package and for the example of MCMC multi-line files posteriors of derived quantities can be plotted.

8. Population command and subcommand syntax

For ease of reading Casal2 files in text editors, there exists a syntax highlighter `CASAL2.syn`

8.1. Model structure

`@model` *`label`* Define an object type Model

`age_plus` Define the oldest age as a plus group
Type: boolean
Default: false
Value: true, false

`final_year` Define the final year of the model, excluding years in the projection period
Type: non-negative integer
Default: No Default
Value: Defines the last year of the model, i.e., the model is run from start_year to final_year

`initialisation_phases` Define the labels of the phases of the initialisation
Type: string vector
Default: true
Value: A list of valid labels defined by `@initialisation_phase`

`label`
Type: string
Default: No Default

`length_bins`
Type: constant vector
Default: true

`max_age` Maximum age of individuals in the population
Type: non-negative integer
Default: 0
Value: $0 \leq \text{age}_{\text{min}} \leq \text{age}_{\text{max}}$

`min_age` Minimum age of individuals in the population
Type: non-negative integer
Default: 0
Value: $0 \leq \text{age}_{\text{min}} \leq \text{age}_{\text{max}}$

`projection_final_year` Define the final year of the model in projection mode
Type: non-negative integer
Default: 0
Value: Defines the last year of the projection period, i.e., the projection period runs from `final_year`+1 to `projection_final_year`. For the default, 0, no projections are run.

`start_year` Define the first year of the model, immediately following initialisation
Type: non-negative integer
Default: No Default
Value: Defines the first year of the model, ≥ 1, e.g. 1990

`time_steps` Define the labels of the time steps, in the order that they are applied, to form the annual cycle
Type: string vector
Default: No Default
Value: A list of valid labels defined by `@time_step`

`type` Type of model (the partition structure). Either age, length or hybrid
Type: string
Default: age

8.2. Initialisation

`@initialisation_phase` *`label`* Define an object type Initialisation_Phase

`label` Label
Type: string
Default: No Default

`type` Type
Type: string
Default: iterative

8.2.1. `@initialisation__phase[label].type=cinitial`

`categories` List of categories to use
Type: string vector
Default: No Default

8.2.2. `@initialisation__phase[label].type=derived`

`casal_intialisation_switch` Reset the partition after running an extra annual cycle to take on equilibrium SSB's. Warning should only be set to true if comparing with previous CASAL models
Type: boolean
Default: false

`exclude_processes` The processes to exclude from all time steps
Type: string vector
Default: true

`insert_processes` The processes to insert in to target time steps
Type: string vector
Default: true

8.2.3. @initialisation_phase[label].type=iterative

`convergence_years` The years to test for convergence
Type: non-negative integer vector
Default: true

`exclude_processes` The processes to exclude from all time steps
Type: string vector
Default: true

`insert_processes` The processes to insert in to target time steps
Type: string vector
Default: true

`lambda` Lambda
Type: constant
Default: Double(0.0

`years` The number of iterations to execute this phase for
Type: non-negative integer
Default: No Default

8.2.4. @initialisation_phase[label].type=state_category_by_age

`categories` List of categories to use
Type: string vector
Default: No Default

`max_age` Maximum age to use for this process
Type: non-negative integer
Default: No Default

`min_age` Minimum age to use for this process
Type: non-negative integer
Default: No Default

8.3. Categories

@categories *label* Define an object type Categories

`age_lengths` The labels of age_length objects that are assigned to categories
Type: string vector
Default: true

`format` The format that the category names should adhere too
Type: string
Default: No Default

`names` The names of the categories to be used in the model
Type: string vector
Default: No Default

`years` The years that individual categories will be active for. This overrides the model values
Type: string vector
Default: true

8.4. Time-steps

@time_step *label* Define an object type Time_Step

`label` Label
Type: string
Default: No Default

`processes` Processes
Type: string vector
Default: No Default

`type`
Type: string
Default: No Default

8.5. Processes

@process *label* Define an object type Process

`print_report` Generate parameter report
Type: boolean
Default: false

`label` Label

Type: string
Default: No Default

`type` Type
Type: string
Default: ””

8.5.1. @process[label].type=ageing

`categories` Categories
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

8.5.2. @process[label].type=growth

`print_report` Generate parameter report
Type: boolean
Default: false

8.5.3. @process[label].type=maturation

`print_report` Generate parameter report
Type: boolean
Default: false

`from` List of categories to mature from
Type: string vector
Default: No Default

`rates` The rates to mature for each year
Type: constant vector
Default: No Default

`selectivities` List of selectivities to use for maturation
Type: string vector
Default: No Default

`to` List of categories to mature too

Type: string vector
Default: No Default

`years` The years to be associated with rates
Type: non-negative integer vector
Default: No Default

8.5.4. `@process[label].type=mortality_constant_rate`

`categories` List of categories
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

`m` Mortality rates
Type: constant vector
Default: No Default

`time_step_ratio` Time step ratios for M
Type: constant vector
Default: true

`selectivities` Selectivities
Type: string vector
Default: No Default

8.5.5. `@process[label].type=mortality_event`

`catches` Catches
Type: constant vector
Default: No Default

`categories` Categories
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

`penalty` Penalty label

Type: string
Default: ""

`selectivities` List of selectivities
Type: string vector
Default: No Default

`u_max` U Max
Type: constant
Default: 0.99

`years` Years
Type: non-negative integer vector
Default: No Default

8.5.6. `@process[label].type=mortality_event_biomass`

`catches` Catches for each year
Type: constant vector
Default: No Default

`categories` Category labels
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

`penalty` Penalty label
Type: string
Default: ""

`selectivities` Selectivity labels
Type: string vector
Default: No Default

`u_max` U Max
Type: constant
Default: 0.99

`units` Unit of weight that the Catches table are expressed in
Type: string
Default: No Default

`years` Years to apply mortality
Type: non-negative integer vector
Default: No Default

8.5.7. @process[label].type=mortality_holling_rate

`a` parameter a
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`b` parameter b
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`print_report` Generate parameter report
Type: boolean
Default: false

`penalty` Label of penalty to be applied
Type: string
Default: ””

`predator_categories` Predator Categories labels
Type: string vector
Default: No Default

`predator_selectivities` Selectivities for predator categories
Type: string vector
Default: No Default

`prey_categories` Prey Categories labels
Type: string vector
Default: No Default

`prey_selectivities` Selectivities for prey categories
Type: string vector
Default: No Default

`u_max` Umax
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`x` parameter x
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

8.5.8. `@process[label].type=mortality_instantaneous`

`categories` Categories for natural mortality
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

`m` Mortality rates
Type: constant vector
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`selectivities` Selectivities for Natural Mortality
Type: string vector
Default: No Default

`time_step_ratio` Time step ratios for M
Type: constant vector
Default: true

`units` Unit of weight that the Catches table are expressed in
Type: string
Default: No Default

8.5.9. @process[label].type=mortality_prey_suitability

`consumption_rate` Predator consumption rate
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`print_report` Generate parameter report
Type: boolean
Default: false

`electivities` Prey Electivities
Type: constant vector
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`penalty` Label of penalty to be applied
Type: string
Default: ""

`predator_categories` Predator Categories labels
Type: string vector
Default: No Default

`predator_selectivities` Selectivities for predator categories
Type: string vector
Default: No Default

`prey_categories` Prey Categories labels
Type: string vector
Default: No Default

`prey_selectivities` Selectivities for prey categories
Type: string vector
Default: No Default

`u_max` Umax
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`years` Year that process occurs

Type: non-negative integer vector
Default: No Default

8.5.10. @process[label].type=nop

print_report Generate parameter report
Type: boolean
Default: false

8.5.11. @process[label].type=recruitment_beverton_holt

age Age to recruit at
Type: non-negative integer
Default: true

b0 B0
Type: constant
Default: false

units Units of B0, if initialising model using B0
Type: string
Default: ""

categories Category labels
Type: string vector
Default: No Default

print_report Generate parameter report
Type: boolean
Default: false

b0_intialisation_phase Initialisation phase Label that b0 is from
Type: string
Default: ""

prior_standardised_ycs Priors for year class strength on ycs values (not standardised ycs values
Type: boolean
Default: true

proportions Proportions
Type: constant vector
Default: No Default

`r0` R0
Type: constant
Default: false

`ssb` SSB Label (derived quantity
Type: string
Default: No Default

`ssb_offset` Spawning biomass year offset
Type: integer
Default: false

`standardise_ycs_years` Years that are included for year class standardisation
Type: non-negative integer vector
Default: true

`steepness` Steepness
Type: constant
Default: 1.0

`ycs_values` YCS Values
Type: constant vector
Default: No Default

8.5.12. @process[label].type=recruitment_constant

`age` Age
Type: non-negative integer
Default: No Default

`categories` Categories
Type: string vector
Default: No Default

`print_report` Generate parameter report
Type: boolean
Default: false

`proportions` Proportions
Type: constant vector
Default: true

`r0` R0

Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

8.5.13. @process[label].type=tag_by_age

`print_report` Generate parameter report
Type: boolean
Default: false

`from` Categories to transition from
Type: string vector
Default: No Default

`initial_mortality`
Type: constant
Default: Double(0

`initial_mortality_selectivity`
Type: string
Default: ""

`loss_rate`
Type: constant vector
Default: No Default

`loss_rate_selectivities`
Type: string vector
Default: true

`max_age` Maximum age to transition
Type: non-negative integer
Default: No Default

`min_age` Minimum age to transition
Type: non-negative integer
Default: No Default

`n`
Type: constant vector
Default: true

`penalty` Penalty label
Type: string
Default: ""

selectivities
Type: string vector
Default: No Default

to Categories to transition to
Type: string vector
Default: No Default

u_max U Max
Type: constant
Default: 0.99

years Years to execute the transition in
Type: non-negative integer vector
Default: No Default

8.5.14. @process[label].type=tag_by_length

print_report Generate parameter report
Type: boolean
Default: false

from Categories to transition from
Type: string vector
Default: No Default

initial_mortality
Type: constant
Default: Double(0

initial_mortality_selectivity
Type: string
Default: ""

maximum_length The upper length when there is no plus group
Type: constant
Default: Double(0

n
Type: constant vector
Default: true

penalty Penalty label

Type: string
Default: ""

plus_group Use plus group for last length bin
Type: boolean
Default: false

selectivities
Type: string vector
Default: No Default

to Categories to transition to
Type: string vector
Default: No Default

u_max U Max
Type: constant
Default: 0.99

years Years to execute the transition in
Type: non-negative integer vector
Default: No Default

8.5.15. @process[label].type=tag_loss

categories List of categories
Type: string vector
Default: No Default

print_report Generate parameter report
Type: boolean
Default: false

time_step_ratio Time step ratios for Tag Loss
Type: constant vector
Default: true

selectivities Selectivities
Type: string vector
Default: No Default

tag_loss_rate Tag Loss rates
Type: constant vector
Default: No Default

`tag_loss_type` Type of tag loss
Type: string
Default: No Default

`year` The year the first tagging release process was executed
Type: non-negative integer
Default: No Default

8.5.16. @process[label].type=transition_category

`print_report` Generate parameter report
Type: boolean
Default: false

`from` From
Type: string vector
Default: No Default

`proportions` Proportions
Type: constant vector
Default: No Default

`selectivities` Selectivity names
Type: string vector
Default: No Default

`to` To
Type: string vector
Default: No Default

8.5.17. @process[label].type=transition_category_by_age

`print_report` Generate parameter report
Type: boolean
Default: false

`from` Categories to transition from
Type: string vector
Default: No Default

`max_age` Maximum age to transition
Type: non-negative integer
Default: No Default

`min_age` Minimum age to transition
Type: non-negative integer
Default: No Default

`penalty` Penalty label
Type: string
Default: ""

`to` Categories to transition to
Type: string vector
Default: No Default

`u_max` U Max
Type: constant
Default: 0.99

`years` Years to execute the transition in
Type: non-negative integer vector
Default: No Default

8.6. Time varying parameters

`@time_varying` *`label`* Define an object type Time_Varying

`label` Label
Type: string
Default: No Default

`parameter` Parameter to vary
Type: string
Default: No Default

`type` Type
Type: string
Default: ""

`years` Years to recalculate the values
Type: non-negative integer vector
Default: No Default

8.6.1. `@time_varying[label].type=annual_shift`

`a`
Type: constant
Default: No Default

`b`
Type: constant
Default: No Default

`c`
Type: constant
Default: No Default

`parameter` Parameter to vary
Type: string
Default: No Default

`scaling_years`
Type: non-negative integer vector
Default: true

`values`
Type: constant vector
Default: No Default

`years` Years to recalculate the values
Type: non-negative integer vector
Default: No Default

8.6.2. @time_varying[label].type=constant

`parameter` Parameter to vary
Type: string
Default: No Default

`value` Value to assign to estimable
Type: constant
Default: No Default

`years` Years to recalculate the values
Type: non-negative integer vector
Default: No Default

8.6.3. @time_varying[label].type=exogenous

`a` Shift parameter
Type: constant
Default: No Default

`exogeneous_variable` Values of exogeneous variable for each year
Type: constant vector
Default: No Default

`parameter` Parameter to vary
Type: string
Default: No Default

`years` Years to recalculate the values
Type: non-negative integer vector
Default: No Default

8.6.4. @time_varying[label].type=random_walk

`distribution` distribution
Type: string
Default: normal

`mean` Mean
Type: constant
Default: 0

`parameter` Parameter to vary
Type: string
Default: No Default

`sigma` Standard deviation
Type: constant
Default: 1

`years` Years to recalculate the values
Type: non-negative integer vector
Default: No Default

8.7. Derived quantities

@derived_quantity *label* Define an object type Derived_Quantity

`categories` The list of categories to use when calculating the derived quantity
Type: string vector
Default: No Default

`label` Label

Type: string
Default: No Default

`time_step_proportion_method`
Type: string
Default: weighted_sum
Allowed Values: weighted_sum, weighted_product

`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
Type: string vector
Default: No Default

`time_step` The time step to calculate the derived quantity after
Type: string
Default: No Default

`time_step_proportion`
Type: constant
Default: Double(1.0

`type` Type
Type: string
Default: No Default

8.7.1. @derived_quantity[label].type=abundance

`categories` The list of categories to use when calculating the derived quantity
Type: string vector
Default: No Default

`time_step_proportion_method`
Type: string
Default: weighted_sum
Allowed Values: weighted_sum, weighted_product

`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
Type: string vector
Default: No Default

`time_step` The time step to calculate the derived quantity after
Type: string
Default: No Default

time_step_proportion
Type: constant
Default: Double(1.0

8.7.2. @derived_quantity[label].type=biomass

categories The list of categories to use when calculating the derived quantity
Type: string vector
Default: No Default

time_step_proportion_method
Type: string
Default: weighted_sum
Allowed Values: weighted_sum, weighted_product

selectivities The list of selectivities to use when calculating the derived quantity. 1 per category
Type: string vector
Default: No Default

time_step The time step to calculate the derived quantity after
Type: string
Default: No Default

time_step_proportion
Type: constant
Default: Double(1.0

8.8. Age-length relationship

@age_length *label* Define an object type Age_Length

casal_switch A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
Type: boolean
Default: false

cv_first CV for the first age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

cv_last CV for last age class

Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`distribution` TBA
Type: string
Default: normal

`label` Label
Type: string
Default: No Default

`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
Type: constant vector
Default: true

`type` Type
Type: string
Default: No Default

8.8.1. @age_length[label].type=data

`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age
Type: boolean
Default: true

`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
Type: boolean
Default: false

`cv_first` CV for the first age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`cv_last` CV for last age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`distribution` TBA

Type: string
Default: normal

`external_gaps`
Type: string
Default: mean
Allowed Values: mean, nearest_neighbour

`internal_gaps`
Type: string
Default: mean
Allowed Values: mean, nearest_neighbour, interpolate

`length_weight` TBA
Type: string
Default: No Default

`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
Type: constant vector
Default: true

8.8.2. @age_length[label].type=none

`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
Type: boolean
Default: false

`cv_first` CV for the first age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`cv_last` CV for last age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`distribution` TBA
Type: string
Default: normal

`time_step_proportions` the proportion increase of age through the in each time step that

corresponds to a length and thus weight increase
Type: constant vector
Default: true

8.8.3. `@age_length[label].type=schnute`

`a` TBA
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`b` TBA
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`by_length` TBA
Type: boolean
Default: true

`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
Type: boolean
Default: false

`cv_first` CV for the first age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`cv_last` CV for last age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`distribution` TBA
Type: string
Default: normal

`length_weight` TBA
Type: string
Default: No Default

`tau1` TBA

Type: constant
Default: No Default

`tau2` TBA
Type: constant
Default: No Default

`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
Type: constant vector
Default: true

`y1` TBA
Type: constant
Default: No Default

`y2` TBA
Type: constant
Default: No Default

8.8.4. `@age_length[label].type=von_bertalanffy`

`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age
Type: boolean
Default: true

`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
Type: boolean
Default: false

`cv_first` CV for the first age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`cv_last` CV for last age class
Type: constant
Default: Double(0.0
Lower Bound: 0.0 (inclusive)

`distribution` TBA
Type: string
Default: normal

`k` TBA
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`length_weight` TBA
Type: string
Default: No Default

`linf` TBA
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`t0` TBA
Type: constant
Default: No Default

`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
Type: constant vector
Default: true

8.9. Length-weight

`@length_weight` *`label`* Define an object type Length_Weight

`label` Label
Type: string
Default: No Default

`type` Type
Type: string
Default: No Default

8.9.1. `@length_weight[label].type=basic`

`a` A
Type: constant
Default: No Default

`b` B
Type: constant
Default: No Default

`units` Units of measure (tonnes, kgs, grams
Type: string
Default: No Default

8.9.2. @length_weight[label].type=none

8.10. Selectivities

@selectivity *label* Define an object type Selectivity

`label` Label
Type: string
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`type` Type
Type: string
Default: No Default

8.10.1. @selectivity[label].type=all_values

`length_based` Is the selectivity length based
Type: boolean
Default: false

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`v` V
Type: constant vector
Default: No Default

8.10.2. @selectivity[label].type=all_values_bounded

`h` H
Type: non-negative integer
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`l` L
Type: non-negative integer
Default: No Default

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`v` V
Type: constant vector
Default: No Default

8.10.3. @selectivity[label].type=constant

`c` C
Type: constant
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

8.10.4. @selectivity[label].type=double_exponential

`alpha` Alpha
Type: constant
Default: 1.0

`length_based` Is the selectivity length based
Type: boolean
Default: false

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`x0` X0
Type: constant
Default: No Default

`x1` X1
Type: constant
Default: No Default

`x2` X2
Type: constant
Default: No Default

`y0` Y0
Type: constant
Default: No Default

`y1` Y1
Type: constant
Default: No Default

`y2` Y2
Type: constant
Default: No Default

8.10.5. @selectivity[label].type=double_normal

`alpha` Alpha
Type: constant
Default: 1.0

`length_based` Is the selectivity length based
Type: boolean
Default: false

`mu` Mu

Type: constant
Default: No Default

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`sigma_l` Sigma L
Type: constant
Default: No Default

`sigma_r` Sigma R
Type: constant
Default: No Default

8.10.6. `@selectivity[label].type=increasing`

`alpha` Alpha
Type: constant
Default: 1.0

`h` High
Type: non-negative integer
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`l` Low
Type: non-negative integer
Default: No Default

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

`v` V
Type: constant vector
Default: No Default

8.10.7. @selectivity[label].type=inverse_logistic

a50 A50
Type: constant
Default: No Default

alpha Alpha
Type: constant
Default: 1.0

ato95 aTo95
Type: constant
Default: No Default

length_based Is the selectivity length based
Type: boolean
Default: false

intervals Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

8.10.8. @selectivity[label].type=knife_edge

alpha Alpha
Type: constant
Default: 1.0

e Edge
Type: constant
Default: No Default

length_based Is the selectivity length based
Type: boolean
Default: false

intervals Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

8.10.9. `@selectivity[label].type=logistic`

`a50` A50
Type: constant
Default: No Default

`alpha` Alpha
Type: constant
Default: 1.0

`ato95` Ato95
Type: constant
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

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

`a50` A50
Type: constant
Default: No Default

`alpha` Alpha
Type: constant
Default: 1.0

`ato95` Ato95
Type: constant
Default: No Default

`h` High
Type: non-negative integer
Default: No Default

`length_based` Is the selectivity length based
Type: boolean
Default: false

`l` Low
Type: non-negative integer
Default: No Default

`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
Type: non-negative integer
Default: 5

9. Estimation command and subcommand syntax

9.1. Estimation methods

`@estimate` *`label`* Define an object type Estimate

`estimation_phase` TBA
Type: non-negative integer
Default: 1u

`label` Label
Type: string
Default: ””

`lower_bound` The lowest value the parameter is allowed to have
Type: constant
Default: No Default

`mcmc` TBA
Type: boolean
Default: false

`parameter` The name of the variable to estimate in the model
Type: string
Default: No Default

`prior` The name of the prior to use for the parameter
Type: string
Default: ””

`same` A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ””

`type` Type

Type: string
Default: No Default

`upper_bound` The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.1. @estimate[label].type=beta

`a` A
Type: constant
Default: No Default

`b` B
Type: constant
Default: No Default

`estimation_phase` TBA
Type: non-negative integer
Default: 1u

`lower_bound` The lowest value the parameter is allowed to have
Type: constant
Default: No Default

`mcmc` TBA
Type: boolean
Default: false

`mu` Mu
Type: constant
Default: No Default

`parameter` The name of the variable to estimate in the model
Type: string
Default: No Default

`prior` The name of the prior to use for the parameter
Type: string
Default: ””

`same` A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ””

`sigma` Sigma
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`upper_bound` The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.2. @estimate[label].type=lognormal

`cv` Cv
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`estimation_phase` TBA
Type: non-negative integer
Default: 1u

`lower_bound` The lowest value the parameter is allowed to have
Type: constant
Default: No Default

`mcmc` TBA
Type: boolean
Default: false

`mu` Mu
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`parameter` The name of the variable to estimate in the model
Type: string
Default: No Default

`prior` The name of the prior to use for the parameter
Type: string
Default: ""

`same` A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

upper_bound The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.3. @estimate[label].type=normal

cv Cv
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

estimation_phase TBA
Type: non-negative integer
Default: 1u

lower_bound The lowest value the parameter is allowed to have
Type: constant
Default: No Default

mcmc TBA
Type: boolean
Default: false

mu Mu
Type: constant
Default: No Default

parameter The name of the variable to estimate in the model
Type: string
Default: No Default

prior The name of the prior to use for the parameter
Type: string
Default: ""

same A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

upper_bound The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.4. @estimate[label].type=normal_by_stdev

`estimation_phase` TBA
Type: non-negative integer
Default: 1u

`lower_bound` The lowest value the parameter is allowed to have
Type: constant
Default: No Default

`mcmc` TBA
Type: boolean
Default: false

`mu` Mu
Type: constant
Default: No Default

`parameter` The name of the variable to estimate in the model
Type: string
Default: No Default

`prior` The name of the prior to use for the parameter
Type: string
Default: ""

`same` A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

`sigma` Sigma
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`upper_bound` The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.5. @estimate[label].type=normal_log

`estimation_phase` TBA
Type: non-negative integer
Default: 1u

lower_bound   The lowest value the parameter is allowed to have
Type: constant
Default: No Default

mcmc   TBA
Type: boolean
Default: false

mu   Mu
Type: constant
Default: No Default

parameter   The name of the variable to estimate in the model
Type: string
Default: No Default

prior   The name of the prior to use for the parameter
Type: string
Default: ""

same   A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

sigma   Sigma
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

upper_bound   The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.6. @estimate[label].type=uniform

estimation_phase   TBA
Type: non-negative integer
Default: 1u

lower_bound   The lowest value the parameter is allowed to have
Type: constant
Default: No Default

mcmc   TBA

Type: boolean
Default: false

parameter The name of the variable to estimate in the model
Type: string
Default: No Default

prior The name of the prior to use for the parameter
Type: string
Default: ""

same A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

upper_bound The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.1.7. @estimate[label].type=uniform_log

estimation_phase TBA
Type: non-negative integer
Default: 1u

lower_bound The lowest value the parameter is allowed to have
Type: constant
Default: No Default

mcmc TBA
Type: boolean
Default: false

parameter The name of the variable to estimate in the model
Type: string
Default: No Default

prior The name of the prior to use for the parameter
Type: string
Default: ""

same A list of parameters that are bound to the value of this estimate
Type: string vector
Default: ""

`upper_bound` The highest value the parameter is allowed to have
Type: constant
Default: No Default

9.2. Point estimation

`@minimiser` *`label`* Define an object type Minimiser

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`label` Label
Type: string
Default: No Default

`type` Type of minimiser to use
Type: string
Default: No Default

9.2.1. `@minimiser[label].type=callback_a_d_o_l_c`

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 0.02

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations
Type: integer
Default: 1000

`step_size` Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.2.2. @minimiser[label].type=engine_a_d_o_l_c

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 0.02

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations
Type: integer
Default: 1000

`step_size` Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.2.3. @minimiser[label].type=f_m_m_a_d_o_l_c

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 0.02

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations
Type: integer
Default: 1000

`step_size` Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.2.4. `@minimiser[label].type=beta_diff`

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 2e-3

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations
Type: integer
Default: 1000

9.2.5. `@minimiser[label].type=c_p_p_a_d`

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

9.2.6. `@minimiser[label].type=call_back_d_e_solver`

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`crossover_probability` TBA
Type: constant
Default: 0.9

`difference_scale` The scale to apply to new solutions when comparing candidates
Type: constant
Default: 0.02

`max_generations` The maximum number of iterations to run
Type: non-negative integer
Default: No Default

`method` The type of candidate generation method to use
Type: string
Default: ""
Value: not_yet_implemented

`population_size` The number of candidate solutions to have in the population
Type: non-negative integer
Default: No Default

`tolerance` The total variance between the population and best candidate before acceptance
Type: constant
Default: 0.01

9.2.7. `@minimiser[label].type=engine_d_e_solver`

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created

Type: boolean
Default: true

`crossover_probability` TBA
Type: constant
Default: 0.9

`difference_scale` The scale to apply to new solutions when comparing candidates
Type: constant
Default: 0.02

`max_generations` The maximum number of iterations to run
Type: non-negative integer
Default: No Default

`method` The type of candidate generation method to use
Type: string
Default: ""
Value: not_yet_implemented

9.2.8. @minimiser[label].type=call_back_d_lib

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

9.2.9. @minimiser[label].type=dummy

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

9.2.10. @minimiser[label].type=callback_gamma_diff

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 0.02

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations
Type: integer
Default: 1000

`step_size` Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.2.11. @minimiser[label].type=engine_gamma_diff

`active` True if this minimiser is active
Type: boolean
Default: false

`covariance` True if a covariance matrix should be created
Type: boolean
Default: true

`tolerance` Tolerance of the gradient for convergence
Type: constant
Default: 0.02

`evaluations` Maximum number of evaluations
Type: integer
Default: 4000

`iterations` Maximum number of iterations

Type: integer
Default: 1000

step_size Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.2.12. @minimiser[label].type=f_m_m_gamma_diff

active True if this minimiser is active
Type: boolean
Default: false

covariance True if a covariance matrix should be created
Type: boolean
Default: true

tolerance Tolerance of the gradient for convergence
Type: constant
Default: 0.02

evaluations Maximum number of evaluations
Type: integer
Default: 4000

iterations Maximum number of iterations
Type: integer
Default: 1000

step_size Minimum Step-size before minimisation fails
Type: constant
Default: 1e-7

9.3. Monte Carlo Markov Chain (MCMC)

@mcmc *label* Define an object type MCMC

active Is this the active MCMC algorithm
Type: boolean
Default: true

label Label
Type: string
Default: No Default

`length` The number of chain links to create
Type: non-negative integer
Default: No Default

`print_default_reports`
Type: boolean
Default: true

`type` Type
Type: string
Default: ""

9.3.1. @m_c_m_c[label].type=independence_metropolis

`active` Is this the active MCMC algorithm
Type: boolean
Default: true

`adapt_stepsize_at` Iterations in the chain to check and resize the MCMC stepsize
Type: non-negative integer vector
Default: true

`correlation_adjustment_diff` TBA
Type: constant
Default: 0.0001

`covariance_adjustment_method` Method for adjusting small variances in the covariance proposal matrix
Type: string
Default: covariance

`df` Degrees of freedom of the multivariate t proposal distribution
Type: non-negative integer
Default: 4

`keep` Spacing between recorded values in the chain
Type: non-negative integer
Default: 1u

`length` The number of chain links to create
Type: non-negative integer
Default: No Default

`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal

distribution
Type: constant
Default: 0.8

`print_default_reports`
Type: boolean
Default: true

`proposal_distribution` The shape of the proposal distribution (either t or normal
Type: string
Default: t

`start` Covariance multiplier for the starting point of the Markov chain
Type: constant
Default: 0.0

`step_size` Initial stepsize (as a multiplier of the approximate covariance matrix
Type: constant
Default: 0.02

9.4. Profiles

`@profile` *`label`* Define an object type Profile

`label` Label
Type: string
Default: ""

`lower_bound` The lower bounds
Type: constant
Default: No Default

`parameter` The system parameter to profile
Type: string
Default: No Default

`steps` The number of steps to take between the lower and upper bound
Type: non-negative integer
Default: No Default

`type`
Type: string
Default: No Default

`upper_bound` The upper bounds

Type: constant
Default: No Default

9.5. Defining catchability constants

@catchability *label* Define an object type Catchability

`label` Label
Type: string
Default: No Default

`type`
Type: string
Default: No Default

9.5.1. @catchability[label].type=free

`q` The catchability amount
Type: constant
Default: No Default

9.6. Defining penalties

@penalty *label* Define an object type Penalty

`label` Label
Type: string
Default: No Default

`type` Type
Type: string
Default: No Default

9.6.1. @penalty[label].type=process

`log_scale` Log scale
Type: boolean
Default: false

`multiplier` Multiplier
Type: constant
Default: 1.0

9.7. Defining priors on parameter ratios, differences and means

@additional_prior *label* Define an object type Additional_Prior

label Label
Type: string
Default: No Default

type Type
Type: string
Default: No Default

9.7.1. @additional_prior[label].type=beta

a A
Type: constant
Default: No Default

b B
Type: constant
Default: No Default

mu Mu
Type: constant
Default: No Default

sigma Sigma
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

9.7.2. @additional_prior[label].type=vector_average

k K Value to use in the calculation
Type: constant
Default: No Default

method What calculation method to use (k, l, m
Type: string
Default: k

multiplier Multiplier for the penalty amount
Type: constant
Default: 1

`parameter` Label of the estimate to generate penalty on
Type: string
Default: No Default

9.7.3. `@additional_prior[label].type=vector_smoothing`

`log_scale` Log scale
Type: boolean
Default: false

`lower_bound` First element to apply the penalty to in the vector
Type: non-negative integer
Default: 0u

`multiplier` Multiplier for the penalty amount
Type: constant
Default: 1

`parameter` Label of the estimate to generate penalty on
Type: string
Default: No Default

`r` Penalty applied to rth differences
Type: non-negative integer
Default: 2u

`upper_bound` Last element to apply the penalty to in the vector
Type: non-negative integer
Default: 0u

10. Observation command and subcommand syntax

10.1. Observation types

The observation types available are,

Observations of proportions of individuals by age class

Observations of proportions of individuals between categories within each age class

Relative and absolute abundance observations

Relative and absolute biomass observations

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

`@observation` *`label`* Define an object type Observation

`categories` Category labels to use
Type: string vector
Default: true

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`label` Label
Type: string
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`type` Type of observation
Type: string
Default: No Default

10.1.1. @observation[label].type=process_abundance

`catchability` Abundance catchability
Type: string
Default: No Default

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta value for error values
Type: constant
Default: Double(1e-10

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`error_value` The error values to use against the observation values
Type: constant vector
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`obs` Observation values
Type: string vector
Default: No Default

`process_error` Process error
Type: constant
Default: Double(0.0

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ””

`time_step` Time step to execute in
Type: string
Default: No Default

`years` Years to execute in
Type: non-negative integer vector
Default: No Default

10.1.2. `@observation[label].type=time_step_abundance`

`catchability` TBA
Type: string
Default: No Default

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta value for error values
Type: constant
Default: Double(1e-10

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`error_value` The error values to use against the observation values
Type: constant vector
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`obs` Observation values
Type: string vector
Default: No Default

`process_error` Process error
Type: constant
Default: Double(0.0

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`years` Years to execute in
Type: non-negative integer vector
Default: No Default

10.1.3. @observation[label].type=process_biomass

`catchability` Catchability of Biomass
Type: string
Default: No Default

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta value for error values
Type: constant
Default: Double(1e-10

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`error_value` The error values to use against the observation values
Type: constant vector
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`obs` Observation values
Type: string vector
Default: No Default

`process_error` Process error
Type: constant
Default: Double(0.0

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`years` Years to execute in
Type: non-negative integer vector
Default: No Default

10.1.4. @observation[label].type=time_step_biomass

`catchability` Catchability of Biomass
Type: string
Default: No Default

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta value for error values
Type: constant
Default: Double(1e-10

`error_value_multiplier` Error value multiplier for likelihood

Type: constant
Default: Double(1.0

`error_value` The error values to use against the observation values
Type: constant vector
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`obs` Observation values
Type: string vector
Default: No Default

`process_error` Process error
Type: constant
Default: Double(0.0

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`years` Years to execute in
Type: non-negative integer vector
Default: No Default

10.1.5. @observation[label].type=process_proportions_at_age

age_plus Use age plus group
Type: boolean
Default: true

ageing_error Label of ageing error to use
Type: string
Default: ""

categories Category labels to use
Type: string vector
Default: true

delta Delta
Type: constant
Default: DELTA

error_value_multiplier Error value multiplier for likelihood
Type: constant
Default: Double(1.0

likelihood_multiplier Likelihood score multiplier
Type: constant
Default: Double(1.0

likelihood Type of likelihood to use
Type: string
Default: No Default

max_age Maximum age
Type: non-negative integer
Default: No Default

min_age Minimum age
Type: non-negative integer
Default: No Default

process_errors Process error
Type: constant vector
Default: true

process Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`tolerance` Tolerance
Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.6. @observation[label].type=time_step_proportions_at_age

`age_plus` Use age plus group
Type: boolean
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ""

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`max_age` Maximum age
Type: non-negative integer
Default: No Default

`min_age` Minimum age
Type: non-negative integer
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ””

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`tolerance` Tolerance
Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.7. @observation[label].type=proportions_at_age_for_fishery

age_plus Use age plus group
Type: boolean
Default: true

ageing_error Label of ageing error to use
Type: string
Default: ""

categories Category labels to use
Type: string vector
Default: true

delta Delta
Type: constant
Default: DELTA

error_value_multiplier Error value multiplier for likelihood
Type: constant
Default: Double(1.0

fishery Label of fishery the observation is from
Type: string vector
Default: ""

likelihood_multiplier Likelihood score multiplier
Type: constant
Default: Double(1.0

likelihood Type of likelihood to use
Type: string
Default: No Default

max_age Maximum age
Type: non-negative integer
Default: No Default

min_age Minimum age
Type: non-negative integer
Default: No Default

process_errors Process error
Type: constant vector
Default: true

`process` Process label
Type: string
Default: No Default

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time steps that the fisheries are in
Type: string vector
Default: No Default

`tolerance` Tolerance
Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.8. @observation[label].type=process_proportions_at_length

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`length_bins` Length bins
Type: constant vector
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.9. @observation[label].type=time_step_proportions_at_length

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta

Type: constant
Default: DELTA

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`length_bins` Length bins
Type: constant vector
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ””

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`tolerance` Tolerance for rescaling proportions

Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.10. @observation[label].type=proportions_at_length_for_fishery

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`fishery` Label of fishery the observation is from
Type: string
Default: ””

`length_bins` Length bins
Type: constant vector
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ””

`time_step` Time step to execute in
Type: string
Default: No Default

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: Double(0.001

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.11. @observation[label].type=process_proportions_by_category

`age_plus` Use age plus group
Type: boolean
Default: true

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA
Lower Bound: 0.0 (exclusive)

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`likelihood_multiplier` Likelihood score multiplier

Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`max_age` Maximum age
Type: non-negative integer
Default: No Default

`min_age` Minimum age
Type: non-negative integer
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`time_step` Time step to execute in

Type: string
Default: No Default

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.12. @observation[label].type=time_step_proportions_by_category

`age_plus` Use age plus group
Type: boolean
Default: true

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA
Lower Bound: 0.0 (exclusive)

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`max_age` Maximum age
Type: non-negative integer
Default: No Default

`min_age` Minimum age
Type: non-negative integer
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.13. @observation[label].type=proportions_migrating

`age_plus` Use age plus group
Type: boolean
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ""

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta

Type: constant
Default: DELTA

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`max_age` Maximum age
Type: non-negative integer
Default: No Default

`min_age` Minimum age
Type: non-negative integer
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`process` Process label
Type: string
Default: No Default

`process_proportion` Process proportion
Type: constant
Default: Double(0.5

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`time_step` Time step to execute in
Type: string
Default: No Default

`years` Year to execute in

Type: non-negative integer vector
Default: No Default

10.1.14. @observation[label].type=tag_recapture_by_age

`age_plus` Use age plus group
Type: boolean
Default: true

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA
Lower Bound: 0.0 (exclusive)

`detection` Detection probability
Type: constant
Default: No Default

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`max_age` Maximum age
Type: non-negative integer
Default: No Default

`min_age` Minimum age
Type: non-negative integer
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: Double(0.5

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.1.15. `@observation[label].type=tag_recapture_by_length`

`categories` Category labels to use
Type: string vector
Default: true

`delta` Delta
Type: constant
Default: DELTA
Lower Bound: 0.0 (exclusive)

`detection` Detection probability
Type: constant
Default: No Default

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: Double(1.0

`length_bins` Length Bins
Type: constant vector
Default: No Default

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: Double(1.0

`likelihood` Type of likelihood to use
Type: string
Default: No Default

`plus_group` Last length bin a plus group
Type: boolean
Default: true

`process_errors` Process error
Type: constant vector
Default: true

`selectivities` Selectivity labels to use
Type: string vector
Default: true

`simulation_likelihood` Simulation likelihood to use
Type: string
Default: ""

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`time_step_proportion` Proportion through the time step to analyse the partition from

Type: constant
Default: Double(0.5

`years` Year to execute in
Type: non-negative integer vector
Default: No Default

10.2. Likelihoods

`@likelihood` *`label`* Define an object type Likelihood

`label`
Type: string
Default: No Default

`type`
Type: string
Default: No Default

10.2.1. @likelihood[label].type=binomial

10.2.2. @likelihood[label].type=binomial_approx

10.2.3. @likelihood[label].type=dirichlet

10.2.4. @likelihood[label].type=log_normal

10.2.5. @likelihood[label].type=log_normal_with_q

10.2.6. @likelihood[label].type=multinomial

10.2.7. @likelihood[label].type=normal

10.2.8. @likelihood[label].type=pseudo

10.3. 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 an object type Ageing_Error

`label` Label
Type: string
Default: No Default

`type` Type
Type: string
Default: No Default

10.3.1. @ageing_error[label].type=data

10.3.2. @ageing_error[label].type=normal

`cv` CV for Misclassification matrix
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`k` TBA
Type: non-negative integer
Default: 0u

10.3.3. @ageing_error[label].type=off_by_one

`k` The minimum age of fish which can be missclassified
Type: non-negative integer
Default: 0u
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`p1` proprtion of misclassification up by a single age, i.e. Proportion of individuals at age 3 that are actually age 4
Type: constant
Default: No Default

`p2` proprtion of misclassification down by a single age
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

11. Report command and subcommand syntax

11.1. Report commands and subcommands

@report *label* Define an object type Report

file_name File Name
Type: string
Default: ""

label Label
Type: string
Default: No Default

type Type
Type: string
Default: No Default

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.1. @report[label].type=ageing_error_matrix

ageing_error Ageing Error label
Type: string
Default: No Default

file_name File Name
Type: string
Default: ""

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.2. @report[label].type=category_info

file_name File Name
Type: string
Default: ""

write_mode Write mode

Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.3. `@report[label].type=category_list`

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.4. `@report[label].type=covariance_matrix`

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.5. `@report[label].type=derived_quantity`

`file_name` File Name
Type: string
Default: ""

`units` Unit of weight output expressed in
Type: string
Default: No Default

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.6. @report[label].type=estimable

`file_name` File Name
Type: string
Default: ””

`parameter` Parameter to print
Type: string
Default: No Default

`time_step` Time Step label
Type: string
Default: ””

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

`years` Years to print the estimable for
Type: non-negative integer vector
Default: No Default

11.1.7. @report[label].type=estimate_summary

`file_name` File Name
Type: string
Default: ””

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.8. @report[label].type=estimate_value

`file_name` File Name
Type: string
Default: ””

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.9. @report[label].type=initialisation_partition

file_name File Name
Type: string
Default: ””

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.10. @report[label].type=mcmc_covariance

file_name File Name
Type: string
Default: ””

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.11. @report[label].type=mcmc_objective

file_name File Name
Type: string
Default: ””

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.12. @report[label].type=mcmc_sample

file_name File Name
Type: string
Default: ””

write_mode Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.13. @report[label].type=m_p_d

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.14. @report[label].type=objective_function

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.15. @report[label].type=observation

`file_name` File Name
Type: string
Default: ""

`observation` Observation label
Type: string
Default: No Default

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.16. @report[label].type=output_parameters

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.17. @report[label].type=partition

`file_name` File Name
Type: string
Default: ""

`time_step` Time Step label
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

`years` Years
Type: non-negative integer vector
Default: true

11.1.18. @report[label].type=partition_biomass

`file_name` File Name
Type: string
Default: ""

`time_step` Time Step label
Type: string
Default: ""

`units` Units (Default Kgs
Type: string
Default: kgs

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

`years` Years

Type: non-negative integer vector
Default: true

11.1.19. @report[label].type=partition_mean_weight

`file_name` File Name
Type: string
Default: ""

`time_step` Time Step label
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

`years` Years
Type: non-negative integer vector
Default: true

11.1.20. @report[label].type=process

`file_name` File Name
Type: string
Default: ""

`process` Process label that is reported
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.21. @report[label].type=project

`file_name` File Name
Type: string
Default: ""

`project` Project label that is reported
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.22. `@report[label].type=random_number_seed`

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.23. `@report[label].type=selectivity`

`file_name` File Name
Type: string
Default: ""

`selectivity` Selectivity name
Type: string
Default: No Default

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.24. `@report[label].type=simulated_observation`

`file_name` File Name
Type: string
Default: ""

`observation` Observation label

Type: string
Default: No Default

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.25. `@report[label].type=standard_header`

`file_name` File Name
Type: string
Default: ""

`write_mode` Write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

12. Other commands and subcommands

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

`file` The name of the external file to include
Type: string
Default: No default
Value: A valid external file
Condition: The file name must be enclosed in double quotes
Example: `@include "my_file.txt"`
Note: `@include` does not denote the end of the previous command block as is the case for all other commands

13. Examples

13.1. An example of a simple model

This example implements a very simple single species and area model, with recruitment, maturation, natural and fishing mortality, and an annual age increment. The population structure has ages $1 - 30^{+}$ with a single category.

Casal2 default file to search for in your current working directory is `casal2.txt`. In this example, `casal2.txt` specifies all the files necessary to run your Casal2 model from your current working directory. This is done using the `!include` command as follows.

```
!include "population.csl2
!include "reports.csl2"
!include "Observation.csl2"
!include "estimation.csl2"
```

Breaking up a Casal2 model into sections is recommended, as it aids in readability and error checking. `population.csl2` contains the population information. The model runs from 1975-2012 and is initialised over a 120 year period prior to 1975, which applies the following processes,

1. A Beverton-Holt recruitment process, recruiting a constant number of individuals to the first age class (i.e., $age = 1$).
2. A constant mortality process representing natural mortality(M). This process is repeated in all three time steps, so that each with its own time step proportion of M applied.
3. An ageing process, where all individuals are aged by one year, and with a plus group accumulator age class at $age = 30$.

Following initialisation, the model runs from the years 1975 to 2012 iterating through two time-steps. The first time-step applies processes of recruitment, and $\frac{1}{2}M_1 + F + \frac{1}{2}M_1$ processes, where M_1 is the proportion of M applied in the first time step. The exploitation process (fishing) is applied in the years 1975–2012. Catches are defined in the catches table and attribute information on each fishery such as selectivity and time-step they are implemented are in the fisheries table in the `@process` block.

The second time-step applies an age increment and the remaining natural mortality.

The first 28 lines of the main section of the `population.csl2` are,

```
## Model Block
@model
start_year 1975
final_year 2012
min_age 1
max_age 30
age_plus true
initialisation_phases iphase1
time_steps step1 step2

## Category Block
@categories
format
names stock
age_lengths age_size

## Initialisation block
@initialisation_phase iphase1
type iterative
years 120

## Annual Cycle definition
@time_step step1
processes Recruitment instant_mort

@time_step step2
processes Ageing instant_mort
```

To carry out a run of the model (to verify that the model runs without any syntax errors), use the command `casal2 -r`. Note that as Casal2 looks for a file named `casal2.txt` by default, we can override this. Hypothetically speaking if our model was all written in `Mymodel.txt` we could call it using the `-c` command like `casal2 -r -c Mymodel.txt`.

To run an estimation, and hence estimate the parameters defined in the file `estimation.csl2` (the catchability constant q, recruitment R_0, and the selectivity parameters a_{50} and a_{to95}), use `casal2 -e`. Here, we have piped the output to `estimate.log` using the command `casal2 -e > estimate.log`, reports the user defined reports `reports.csl2` from the final iteration of the estimation, and successful convergence printed to screen,

```
Total elapsed time: 1 second
Completed
```

The main part of the output from the estimation run is summarised in the file `estimate.log`, and the final MPD parameter values can be piped out as a separate report, in this case named `paramaters.out`, using the command `casal2 -e -o paramaters.out > estimate.log`.

A profile on the R_0 parameter can also be run, using `casal2 -p > profile.log`. See the examples folder for an example of the output.

KATH note below, will be useful to copy that document across. Examples on Input file specification go to the file Input File Specification.odt found in CASAL2/Documentation/Software Development

13.2. In line declaration

In line declarations can help shorten models by by passing `@` blocks, for example

```
@observation chatCPUE
type biomass
```

```
catchability [q=6.52606e-005]
time_step one
categories male+female
selectivities chatFselMale chatFselFemale
likelihood lognormal
years 1992:2001
time_step_proportion 1.0
obs 1.50 1.10 0.93 1.33 1.53 0.90 0.68 0.75 0.57 1.23
error_value 0.35

@estimate
parameter catchability[chatTANbiomass.one].q
type uniform_log
lower_bound 1e-2
upper_bound 1
In line declaration tips
```

In the above code we are defining and estimating catchabbility without explicitely creating an `@catchability` block

When you do an inline declaration the new object will be created with the name of the creator's label.<index> where index will be the word if it's one-nine and the number if it's 10+, for example

```
@mortality halfm
selectivities [type=constant; c=1]

would create
@selectivity halfm.one
```

if there were 10 categories all with there own selectivity the $10^t h$ selectivity would be labelled;

```
@selectivity halfm.10
```

14. Post processing output using R

In the downloaded bundle is a R-package that reads Casal2 output into R. The Casal2 package has only one function `extract()`, which will read in the entire file. The reporting framework is set up so the each `@report` will start with * and end with *end. If this is not the case the `extract()` function will most likely fail. A post processing package is being developed to then create plots and process the raw input from the `extract()` function.

15. Troubleshooting

15.1. Introduction

15.2. Reporting errors

When reporting a bug or problem to the Casal2 development team at casal2@niwa.co.nz, please address the following points.

15.3. Guidelines for reporting a problem with Casal2

1. Detail the version of Casal2 are you using? e.g., Casal2 2016-05-22 (rev. 2271ffd)Microsoft Windows executable"
2. What operating system or environment are you using? e.g., "IBM-PC Intel CPU running Microsoft Windows 8.1 Enterprise, Service Pack 1".
3. Give a brief one-line description of the problem, e.g., "a segmentation fault was reported".
4. If the problem is reproducible, please list the exact steps required to cause it, remembering to include the relevant Casal2 configuration file, other input files, and any out generated. Specify the *exact* command line arguments that were used, e.g., "Using the command `***.-* -*` 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., "Casal2 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 is usually not possible to determine the cause of the problem, and we are unlikely to provide any assistance. Note that it is helpful to be as specific as possible when describing the problem.

16. Acknowledgements

17. Quick reference

@ageing_error *label* Define an object type Ageing_Error

label Label
type Type

@ageing__error[label].type=data

@ageing__error[label].type=normal

cv CV for Misclassification matrix
k TBA

@ageing__error[label].type=off_by_one

k The minimum age of fish which can be missclassified
p1 proprtion of misclassification up by a single age, i.e. Proportion of individuals at age 3 that are actually age 4
p2 proprtion of misclassification down by a single age
@age_length *label* Define an object type Age_Length

casal_switch A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
cv_first CV for the first age class
cv_last CV for last age class
distribution TBA
label Label
time_step_proportions the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
type Type

@age__length[label].type=data

by_length Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age
casal_switch A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
cv_first CV for the first age class
cv_last CV for last age class
distribution TBA
external_gaps
internal_gaps
length_weight TBA
time_step_proportions the proportion increase of age through the in each time step that corresponds to a length and thus weight increase

@age__length[label].type=none

casal_switch A switch to use CASAL Cumulative normal function, note CASAL2 uses the

recent BOOST function which differs from the previous CASAL algorithm
`cv_first` CV for the first age class
`cv_last` CV for last age class
`distribution` TBA
`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase

@age_length[label].type=schnute

`a` TBA
`b` TBA
`by_length` TBA
`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
`cv_first` CV for the first age class
`cv_last` CV for last age class
`distribution` TBA
`length_weight` TBA
`tau1` TBA
`tau2` TBA
`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase
`y1` TBA
`y2` TBA

@age_length[label].type=von_bertalanffy

`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age
`casal_switch` A switch to use CASAL Cumulative normal function, note CASAL2 uses the recent BOOST function which differs from the previous CASAL algorithm
`cv_first` CV for the first age class
`cv_last` CV for last age class
`distribution` TBA
`k` TBA
`length_weight` TBA
`linf` TBA
`t0` TBA
`time_step_proportions` the proportion increase of age through the in each time step that corresponds to a length and thus weight increase

@catchability *label* Define an object type Catchability

`label` Label
`type`

@catchability[label].type=free

`q` The catchability amount

@derived_quantity *label* Define an object type Derived_Quantity

`categories` The list of categories to use when calculating the derived quantity

`label` Label
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`
`type` Type

@derived_quantity[label].type=abundance

`categories` The list of categories to use when calculating the derived quantity
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`

@derived_quantity[label].type=biomass

`categories` The list of categories to use when calculating the derived quantity
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`
@estimate *label* Define an object type Estimate

`estimation_phase` TBA
`label` Label
`lower_bound` The lowest value the parameter is allowed to have
`mcmc` TBA
`parameter` The name of the variable to estimate in the model
`prior` The name of the prior to use for the parameter
`same` A list of parameters that are bound to the value of this estimate
`type` Type
`upper_bound` The highest value the parameter is allowed to have

@estimate[label].type=beta

`a` A
`b` B
`estimation_phase` TBA
`lower_bound` The lowest value the parameter is allowed to have
`mcmc` TBA
`mu` Mu
`parameter` The name of the variable to estimate in the model
`prior` The name of the prior to use for the parameter
`same` A list of parameters that are bound to the value of this estimate
`sigma` Sigma
`upper_bound` The highest value the parameter is allowed to have

@estimate[label].type=lognormal

cv Cv
estimation_phase TBA
lower_bound The lowest value the parameter is allowed to have
mcmc TBA
mu Mu
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
upper_bound The highest value the parameter is allowed to have

@estimate[label].type=normal

cv Cv
estimation_phase TBA
lower_bound The lowest value the parameter is allowed to have
mcmc TBA
mu Mu
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
upper_bound The highest value the parameter is allowed to have

@estimate[label].type=normal_by_stdev

estimation_phase TBA
lower_bound The lowest value the parameter is allowed to have
mcmc TBA
mu Mu
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
sigma Sigma
upper_bound The highest value the parameter is allowed to have

@estimate[label].type=normal_log

estimation_phase TBA
lower_bound The lowest value the parameter is allowed to have
mcmc TBA
mu Mu
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
sigma Sigma
upper_bound The highest value the parameter is allowed to have

@estimate[label].type=uniform

estimation_phase TBA

lower_bound The lowest value the parameter is allowed to have
mcmc TBA
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
upper_bound The highest value the parameter is allowed to have

@estimate[label].type=uniform_log

estimation_phase TBA
lower_bound The lowest value the parameter is allowed to have
mcmc TBA
parameter The name of the variable to estimate in the model
prior The name of the prior to use for the parameter
same A list of parameters that are bound to the value of this estimate
upper_bound The highest value the parameter is allowed to have

@initialisation_phase *label* Define an object type Initialisation_Phase

label Label
type Type

@initialisation_phase[label].type=cinitial

categories List of categories to use

@initialisation_phase[label].type=derived

casal_intialisation_switch Reset the partition after running an extra annual cycle to take on equilibrium SSB's. Warning should only be set to true if comparing with previous CASAL models
exclude_processes The processes to exclude from all time steps
insert_processes The processes to insert in to target time steps

@initialisation_phase[label].type=iterative

convergence_years The years to test for convergence
exclude_processes The processes to exclude from all time steps
insert_processes The processes to insert in to target time steps
lambda Lambda
years The number of iterations to execute this phase for

@initialisation_phase[label].type=state_category_by_age

categories List of categories to use
max_age Maximum age to use for this process
min_age Minimum age to use for this process

@likelihood *label* Define an object type Likelihood

label
type

@likelihood[label].type=binomial

@likelihood[label].type=binomial_approx

@likelihood[label].type=dirichlet

@likelihood[label].type=log_normal

@likelihood[label].type=log_normal_with_q

@likelihood[label].type=multinomial

@likelihood[label].type=normal

@likelihood[label].type=pseudo

@derived_quantity *label* Define an object type Derived_Quantity

`categories` The list of categories to use when calculating the derived quantity
`label` Label
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`
`type` Type

@derived__quantity[label].type=abundance

`categories` The list of categories to use when calculating the derived quantity
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`

@derived__quantity[label].type=biomass

`categories` The list of categories to use when calculating the derived quantity
`time_step_proportion_method`
`selectivities` The list of selectivities to use when calculating the derived quantity. 1 per category
`time_step` The time step to calculate the derived quantity after
`time_step_proportion`

@mcmc *label* Define an object type MCMC

`active` Is this the active MCMC algorithm
`label` Label
`length` The number of chain links to create
`print_default_reports`
`type` Type

@m_c_m_c[label].type=independence_metropolis

`active` Is this the active MCMC algorithm
`adapt_stepsize_at` Iterations in the chain to check and resize the MCMC stepsize
`correlation_adjustment_diff` TBA
`covariance_adjustment_method` Method for adjusting small variances in the covariance

proposal matrix
`df` Degrees of freedom of the multivariate t proposal distribution
`keep` Spacing between recorded values in the chain
`length` The number of chain links to create
`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
`print_default_reports`
`proposal_distribution` The shape of the proposal distribution (either t or normal
`start` Covariance multiplier for the starting point of the Markov chain
`step_size` Initial stepsize (as a multiplier of the approximate covariance matrix
`@minimiser` *`label`* Define an object type Minimiser

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`label` Label
`type` Type of minimiser to use

`@minimiser[label].type=callback_a_d_o_l_c`

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails

`@minimiser[label].type=engine_a_d_o_l_c`

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails

`@minimiser[label].type=f_m_m_a_d_o_l_c`

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails

`@minimiser[label].type=beta_diff`

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations

@minimiser[label].type=c_p_p_a_d

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created

@minimiser[label].type=call_back_d_e_solver

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`crossover_probability` TBA
`difference_scale` The scale to apply to new solutions when comparing candidates
`max_generations` The maximum number of iterations to run
`method` The type of candidate generation method to use
`population_size` The number of candidate solutions to have in the population
`tolerance` The total variance between the population and best candidate before acceptance

@minimiser[label].type=engine_d_e_solver

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`crossover_probability` TBA
`difference_scale` The scale to apply to new solutions when comparing candidates
`max_generations` The maximum number of iterations to run
`method` The type of candidate generation method to use

@minimiser[label].type=call_back_d_lib

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created

@minimiser[label].type=dummy

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created

@minimiser[label].type=callback_gamma_diff

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails

@minimiser[label].type=engine_gamma_diff

`active` True if this minimiser is active

`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails

`@minimiser[label].type=f_m_m_gamma_diff`

`active` True if this minimiser is active
`covariance` True if a covariance matrix should be created
`tolerance` Tolerance of the gradient for convergence
`evaluations` Maximum number of evaluations
`iterations` Maximum number of iterations
`step_size` Minimum Step-size before minimisation fails
`@model` *`label`* Define an object type Model

`age_plus` Define the oldest age as a plus group
`final_year` Define the final year of the model, excluding years in the projection period
`initialisation_phases` Define the labels of the phases of the initialisation
`label`
`length_bins`
`max_age` Maximum age of individuals in the population
`min_age` Minimum age of individuals in the population
`projection_final_year` Define the final year of the model in projection mode
`start_year` Define the first year of the model, immediately following initialisation
`time_steps` Define the labels of the time steps, in the order that they are applied, to form the annual cycle
`type` Type of model (the partition structure). Either age, length or hybrid
`@observation` *`label`* Define an object type Observation

`categories` Category labels to use
`error_value_multiplier` Error value multiplier for likelihood
`label` Label
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`simulation_likelihood` Simulation likelihood to use
`type` Type of observation

`@observation[label].type=process_abundance`

`catchability` Abundance catchability

categories Category labels to use
delta Delta value for error values
error_value_multiplier Error value multiplier for likelihood
error_value The error values to use against the observation values
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
obs Observation values
process_error Process error
process Process label
process_proportion Process proportion
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
years Years to execute in

@observation[label].type=time_step_abundance

catchability TBA
categories Category labels to use
delta Delta value for error values
error_value_multiplier Error value multiplier for likelihood
error_value The error values to use against the observation values
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
obs Observation values
process_error Process error
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
time_step_proportion Proportion through the time step to analyse the partition from
years Years to execute in

@observation[label].type=process_biomass

catchability Catchability of Biomass
categories Category labels to use
delta Delta value for error values
error_value_multiplier Error value multiplier for likelihood
error_value The error values to use against the observation values
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
obs Observation values
process_error Process error
process Process label
process_proportion Process proportion
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
years Years to execute in

@observation[label].type=time_step_biomass

`catchability` Catchability of Biomass
`categories` Category labels to use
`delta` Delta value for error values
`error_value_multiplier` Error value multiplier for likelihood
`error_value` The error values to use against the observation values
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`obs` Observation values
`process_error` Process error
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`time_step` Time step to execute in
`time_step_proportion` Proportion through the time step to analyse the partition from
`years` Years to execute in

@observation[label].type=process_proportions_at_age

`age_plus` Use age plus group
`ageing_error` Label of ageing error to use
`categories` Category labels to use
`delta` Delta
`error_value_multiplier` Error value multiplier for likelihood
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`max_age` Maximum age
`min_age` Minimum age
`process_errors` Process error
`process` Process label
`process_proportion` Process proportion
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`time_step` Time step to execute in
`tolerance` Tolerance
`years` Year to execute in

@observation[label].type=time_step_proportions_at_age

`age_plus` Use age plus group

ageing_error Label of ageing error to use
categories Category labels to use
delta Delta
error_value_multiplier Error value multiplier for likelihood
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
max_age Maximum age
min_age Minimum age
process_errors Process error
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
time_step_proportion Proportion through the time step to analyse the partition from
tolerance Tolerance
years Year to execute in

@observation[label].type=proportions_at_age_for_fishery

age_plus Use age plus group
ageing_error Label of ageing error to use
categories Category labels to use
delta Delta
error_value_multiplier Error value multiplier for likelihood
fishery Label of fishery the observation is from
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
max_age Maximum age
min_age Minimum age
process_errors Process error
process Process label
simulation_likelihood Simulation likelihood to use
time_step Time steps that the fisheries are in
tolerance Tolerance
years Year to execute in

@observation[label].type=process_proportions_at_length

categories Category labels to use

delta Delta
error_value_multiplier Error value multiplier for likelihood
length_bins Length bins
length_plus_group Is the last bin a plus group
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
process_errors Process error
process Process label
process_proportion Process proportion
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
tolerance Tolerance for rescaling proportions
years Year to execute in

@observation[label].type=time_step_proportions_at_length

categories Category labels to use
delta Delta
error_value_multiplier Error value multiplier for likelihood
length_bins Length bins
length_plus_group Is the last bin a plus group
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
process_errors Process error
selectivities Selectivity labels to use
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
time_step_proportion Proportion through the time step to analyse the partition from
tolerance Tolerance for rescaling proportions
years Year to execute in

@observation[label].type=proportions_at_length_for_fishery

categories Category labels to use
delta Delta
error_value_multiplier Error value multiplier for likelihood
fishery Label of fishery the observation is from
length_bins Length bins
length_plus_group Is the last bin a plus group
likelihood_multiplier Likelihood score multiplier
likelihood Type of likelihood to use
process_errors Process error
process Process label
process_proportion Process proportion
simulation_likelihood Simulation likelihood to use
time_step Time step to execute in
tolerance Tolerance for rescaling proportions
years Year to execute in

@observation[label].type=process_proportions_by_category

`age_plus` Use age plus group
`categories` Category labels to use
`delta` Delta
`error_value_multiplier` Error value multiplier for likelihood
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`max_age` Maximum age
`min_age` Minimum age
`process_errors` Process error
`process` Process label
`process_proportion` Process proportion
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`categories2` Target Categories
`selectivities2` Target Selectivities
`time_step` Time step to execute in
`years` Year to execute in

@observation[label].type=time_step_proportions_by_category

`age_plus` Use age plus group
`categories` Category labels to use
`delta` Delta
`error_value_multiplier` Error value multiplier for likelihood
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`max_age` Maximum age
`min_age` Minimum age
`process_errors` Process error
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`categories2` Target Categories
`selectivities2` Target Selectivities
`time_step` Time step to execute in
`time_step_proportion` Proportion through the time step to analyse the partition from
`years` Year to execute in

@observation[label].type=proportions_migrating

`age_plus` Use age plus group

`ageing_error` Label of ageing error to use
`categories` Category labels to use
`delta` Delta
`error_value_multiplier` Error value multiplier for likelihood
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`max_age` Maximum age
`min_age` Minimum age
`process_errors` Process error
`process` Process label
`process_proportion` Process proportion
`simulation_likelihood` Simulation likelihood to use
`time_step` Time step to execute in
`years` Year to execute in

@observation[label].type=tag_recapture_by_age

`age_plus` Use age plus group
`categories` Category labels to use
`delta` Delta
`detection` Detection probability
`error_value_multiplier` Error value multiplier for likelihood
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`max_age` Maximum age
`min_age` Minimum age
`process_errors` Process error
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`categories2` Target Categories
`selectivities2` Target Selectivities
`time_step` Time step to execute in
`time_step_proportion` Proportion through the time step to analyse the partition from
`years` Year to execute in

@observation[label].type=tag_recapture_by_length

`categories` Category labels to use

`delta` Delta
`detection` Detection probability
`error_value_multiplier` Error value multiplier for likelihood
`length_bins` Length Bins
`likelihood_multiplier` Likelihood score multiplier
`likelihood` Type of likelihood to use
`plus_group` Last length bin a plus group
`process_errors` Process error
`selectivities` Selectivity labels to use
`simulation_likelihood` Simulation likelihood to use
`categories2` Target Categories
`selectivities2` Target Selectivities
`time_step` Time step to execute in
`time_step_proportion` Proportion through the time step to analyse the partition from
`years` Year to execute in

`@penalty` *`label`* Define an object type Penalty

`label` Label
`type` Type

@penalty[label].type=process

`log_scale` Log scale
`multiplier` Multiplier

`@process` *`label`* Define an object type Process

`print_report` Generate parameter report
`label` Label
`type` Type

@process[label].type=ageing

`categories` Categories
`print_report` Generate parameter report

@process[label].type=growth

`print_report` Generate parameter report

@process[label].type=maturation

`print_report` Generate parameter report
`from` List of categories to mature from
`rates` The rates to mature for each year
`selectivities` List of selectivities to use for maturation
`to` List of categories to mature too
`years` The years to be associated with rates

@process[label].type=mortality_constant_rate

`categories` List of categories

print_report Generate parameter report
m Mortality rates
time_step_ratio Time step ratios for M
selectivities Selectivities

@process[label].type=mortality_event

catches Catches
categories Categories
print_report Generate parameter report
penalty Penalty label
selectivities List of selectivities
u_max U Max
years Years

@process[label].type=mortality_event_biomass

catches Catches for each year
categories Category labels
print_report Generate parameter report
penalty Penalty label
selectivities Selectivity labels
u_max U Max
units Unit of weight that the Catches table are expressed in
years Years to apply mortality

@process[label].type=mortality_holling_rate

a parameter a
b parameter b
print_report Generate parameter report
penalty Label of penalty to be applied
predator_categories Predator Categories labels
predator_selectivities Selectivities for predator categories
prey_categories Prey Categories labels
prey_selectivities Selectivities for prey categories
u_max Umax
x parameter x
years Year to execute in

@process[label].type=mortality_instantaneous

categories Categories for natural mortality
print_report Generate parameter report
m Mortality rates
selectivities Selectivities for Natural Mortality
time_step_ratio Time step ratios for M
units Unit of weight that the Catches table are expressed in

@process[label].type=mortality_prey_suitability

`consumption_rate` Predator consumption rate
`print_report` Generate parameter report
`electivities` Prey Electivities
`penalty` Label of penalty to be applied
`predator_categories` Predator Categories labels
`predator_selectivities` Selectivities for predator categories
`prey_categories` Prey Categories labels
`prey_selectivities` Selectivities for prey categories
`u_max` Umax
`years` Year that process occurs

@process[label].type=nop

`print_report` Generate parameter report

@process[label].type=recruitment_beverton_holt

`age` Age to recruit at
`b0` B0
`units` Units of B0, if initialising model using B0
`categories` Category labels
`print_report` Generate parameter report
`b0_intialisation_phase` Initialisation phase Label that b0 is from
`prior_standardised_ycs` Priors for year class strength on ycs values (not standardised ycs values
`proportions` Proportions
`r0` R0
`ssb` SSB Label (derived quantity
`ssb_offset` Spawning biomass year offset
`standardise_ycs_years` Years that are included for year class standardisation
`steepness` Steepness
`ycs_values` YCS Values

@process[label].type=recruitment_constant

`age` Age
`categories` Categories
`print_report` Generate parameter report
`proportions` Proportions
`r0` R0

@process[label].type=tag_by_age

`print_report` Generate parameter report

`from` Categories to transition from
`initial_mortality`
`initial_mortality_selectivity`
`loss_rate`
`loss_rate_selectivities`
`max_age` Maximum age to transition
`min_age` Minimum age to transition
`n`
`penalty` Penalty label
`selectivities`
`to` Categories to transition to
`u_max` U Max
`years` Years to execute the transition in

@process[label].type=tag_by_length

`print_report` Generate parameter report
`from` Categories to transition from
`initial_mortality`
`initial_mortality_selectivity`
`maximum_length` The upper length when there is no plus group
`n`
`penalty` Penalty label
`plus_group` Use plus group for last length bin
`selectivities`
`to` Categories to transition to
`u_max` U Max
`years` Years to execute the transition in

@process[label].type=tag_loss

`categories` List of categories
`print_report` Generate parameter report
`time_step_ratio` Time step ratios for Tag Loss
`selectivities` Selectivities
`tag_loss_rate` Tag Loss rates
`tag_loss_type` Type of tag loss
`year` The year the first tagging release process was executed

@process[label].type=transition_category

`print_report` Generate parameter report
`from` From
`proportions` Proportions
`selectivities` Selectivity names
`to` To

@process[label].type=transition_category_by_age

`print_report` Generate parameter report

`from` Categories to transition from
`max_age` Maximum age to transition
`min_age` Minimum age to transition
`penalty` Penalty label
`to` Categories to transition to
`u_max` U Max
`years` Years to execute the transition in

`@profile` *`label`* Define an object type Profile

`label` Label
`lower_bound` The lower bounds
`parameter` The system parameter to profile
`steps` The number of steps to take between the lower and upper bound
`type`
`upper_bound` The upper bounds

`@report` *`label`* Define an object type Report

`file_name` File Name
`label` Label
`type` Type
`write_mode` Write mode

`@report[label].type=ageing_error_matrix`

`ageing_error` Ageing Error label
`file_name` File Name
`write_mode` Write mode

`@report[label].type=category_info`

`file_name` File Name
`write_mode` Write mode

`@report[label].type=category_list`

`file_name` File Name
`write_mode` Write mode

`@report[label].type=covariance_matrix`

`file_name` File Name
`write_mode` Write mode

`@report[label].type=derived_quantity`

`file_name` File Name
`units` Unit of weight output expressed in
`write_mode` Write mode

`@report[label].type=estimable`

`file_name` File Name

parameter Parameter to print
time_step Time Step label
write_mode Write mode
years Years to print the estimable for

@report[label].type=estimate_summary

file_name File Name
write_mode Write mode

@report[label].type=estimate_value

file_name File Name
write_mode Write mode

@report[label].type=initialisation_partition

file_name File Name
write_mode Write mode

@report[label].type=mcmc_covariance

file_name File Name
write_mode Write mode

@report[label].type=mcmc_objective

file_name File Name
write_mode Write mode

@report[label].type=mcmc_sample

file_name File Name
write_mode Write mode

@report[label].type=m_p_d

file_name File Name
write_mode Write mode

@report[label].type=objective_function

file_name File Name
write_mode Write mode

@report[label].type=observation

file_name File Name
observation Observation label
write_mode Write mode

@report[label].type=output_parameters

file_name File Name
write_mode Write mode

@report[label].type=partition

file_name File Name
time_step Time Step label
write_mode Write mode
years Years

@report[label].type=partition_biomass

file_name File Name
time_step Time Step label
units Units (Default Kgs
write_mode Write mode
years Years

@report[label].type=partition_mean_weight

file_name File Name
time_step Time Step label
write_mode Write mode
years Years

@report[label].type=process

file_name File Name
process Process label that is reported
write_mode Write mode

@report[label].type=project

file_name File Name
project Project label that is reported
write_mode Write mode

@report[label].type=random_number_seed

file_name File Name
write_mode Write mode

@report[label].type=selectivity

file_name File Name
selectivity Selectivity name
write_mode Write mode

@report[label].type=simulated_observation

`file_name` File Name
`observation` Observation label
`write_mode` Write mode

`@report[label].type=standard_header`

`file_name` File Name
`write_mode` Write mode
`@selectivity` *`label`* Define an object type Selectivity

`label` Label
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`type` Type

`@selectivity[label].type=all_values`

`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`v` V

`@selectivity[label].type=all_values_bounded`

`h` H
`length_based` Is the selectivity length based
`l` L
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`v` V

`@selectivity[label].type=constant`

`c` C
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution

`@selectivity[label].type=double_exponential`

`alpha` Alpha
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`x0` X0
`x1` X1
`x2` X2
`y0` Y0
`y1` Y1
`y2` Y2

@selectivity[label].type=double_normal

`alpha` Alpha
`length_based` Is the selectivity length based
`mu` Mu
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`sigma_l` Sigma L
`sigma_r` Sigma R

@selectivity[label].type=increasing

`alpha` Alpha
`h` High
`length_based` Is the selectivity length based
`l` Low
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution
`v` V

@selectivity[label].type=inverse_logistic

`a50` A50
`alpha` Alpha
`ato95` aTo95
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution

@selectivity[label].type=knife_edge

`alpha` Alpha
`e` Edge
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution

@selectivity[label].type=logistic

`a50` A50
`alpha` Alpha
`ato95` Ato95
`length_based` Is the selectivity length based
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution

@selectivity[label].type=logistic_producing

`a50` A50

`alpha` Alpha
`ato95` Ato95
`h` High
`length_based` Is the selectivity length based
`l` Low
`intervals` Number of quantiles to evaluate a length based selectivity over the age length distribution

`@length_weight` *`label`* Define an object type Length_Weight

`label` Label
`type` Type

`@length_weight[label].type=basic`

`a` A
`b` B
`units` Units of measure (tonnes, kgs, grams

`@length_weight[label].type=none`

`@time_step` *`label`* Define an object type Time_Step

`label` Label
`processes` Processes
`type`

18. References

R J H Beverton and S J Holt. *On the dynamics of exploited fish populations*. Fishery investigations. HMSO, London, 1957.

B Bull, R I C C Francis, A. Dunn, A McKenzie, D J Gilbert, M H Smith, R Bian, and D Fu. CASAL C++ Algorithmic Stock Assessment Laboratory): CASAL user manual v2.30-2012/03/21. Technical Report 135, NIWA, 2012.

J E Dennis Jr and R B Schnabel. *Numerical methods for unconstrained optimisation and nonlinear equations*. Classics in Applied Mathematics. Prentice Hall, 1996.

A Dunn and SM Hanchet. Southern blue whiting (*Micromesistius australis*) stock assessment for the bounty platform up to, and including, the 2011 season. *New Zealand Fisheries Assessment Report*, 55, 2015.

R I C C Francis, V Haist, and B Bull. Assessment of hoki (macruronus novaezelandiae) in 2002 using a new model. *New Zealand Fisheries Assessment Report*, 6, 2003.

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.

Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10: 1755–1758, 2009.

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`.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1): 25–57, 2006.

Andrea Walther, Andreas Kowarz, and Andreas Griewank. Adol-c: a package for the automatic differentiation of algorithms written in c/c++. *ACM TOMS*, 22(2):131–167, 1996.

19. Casal2 license (GNU GENERAL PUBLIC LICENSE)

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

20. Index