CASAL2 User Manual

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

NIWA Technical Report 139
ISSN 1174-2631
2017

CASAL2 User Manual (modified 2017-07-03) for use with
`casal2-v2017-07-03 (rev. 4480b20e)`

Citation: S. Rasmussen, I. Doonan, A. Dunn, C. Marsh, K. Large, S. Mormede (2017). CASAL2 User Manual, v2017-07-03 (rev. 4480b20e). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 139*. 197 p.

Contents

1. Introduction

CASAL2 is a generalised age-structured population dynamics modelling software package that allows flexibility in specifying model structure, population dynamics, parameter estimation, and model outputs. CASAL2 can model population dynamics for an age-structured population using a range of population dynamics observations, including mark-recapture, relative and absolute abundance time series, and age frequency data. CASAL2 does this by implementing an age-structured population dynamics model that can have user defined categories (e.g., immature, mature, male, female, predator, prey, etc.,) to specify the population structure, and a user-defined age range.

1.1. Where to get CASAL2

In the first instance, see `http://www.niwa.co.nz/` for information about CASAL2. The CASAL2 source code is hosted on github, and can be found at `https://github.com/NIWAFisheriesModelling/CASAL2`.

A Microsoft Windows bundle includes the binary, manual, examples and other help guides. It can be downloaded at `ftp://ftp.niwa.co.nz/Casal2/windows/Casal2.zip` for the Microsoft Windows version. The Linux bundle which includes a binary, manual, examples and other help guides can be downloaded at `ftp://ftp.niwa.co.nz/Casal2/linux/Casal2.tar.gz`.

1.2. 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 undertake an MCMC estimate.

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 occasionally be required.

1.3. Necessary files

For both 64-bit Linux and Microsoft Windows, only the binary executable `casal2` or `casal2.exe` is required to run CASAL2. No other software is required. We do not provide a version for 32-bit operating systems.

CASAL2 offers little in the way of post-processing of model output, and a package available that allows tabulation and graphing of model outputs is recommended. We suggest software such as **R** (R Core Team, 2014) 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).

1.4. Getting help

CASAL2 is distributed as unsupported software, however the Development Team would appreciate being notified of any problems or errors in CASAL2. See Section 15.2 for the recommended template for reporting issues. For further information on CASAL2 please contact the Development Team at casal2@niwa.co.nz.

1.5. Technical details

CASAL2 was compiled on Linux using `gcc` (`http://gcc.gnu.org`), the C/C++ compiler developed by the GNU Project (`http://gcc.gnu.org`). The 64-bit Linux version was compiled using `gcc` version 5.2.1 20151010 Ubuntu Linux (`http://www.ubuntu.com/`). The Microsoft Windows (`http://www.microsoft.com`) version was compiled using MingW (`http://www.mingw.org`) `gcc` (tdm64-1) 5.1.0 (`http://gcc.gnu.org`). The Microsoft Windows(`http://www.microsoft.com`) installer was built using the Inno Setup 5 (`http://www.jrsoftware.org/isdl.php`).

CASAL2 includes number of different minimisers — Different minimisers may be better at some models than others. The first three are non-differentiation based minimisers: the first is closely based on the main algorithm of Dennis Jr and Schnabel (1996), and which uses finite difference gradients; 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.; and the third is Dlib (King, 2009). The three differentiation based minimisers are: ADOLC, an auto differentiation minimiser (Walther et al., 1996); CPPAD an auto differentiation minimiser similar to ADOLC (Wächter and Biegler, 2006); and the third is a modified version of an older version of ADOL-C (v1.8.4) that was used as the auto differentiation minimiser in the first version of CASAL (Bull et al., 2012).

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/NIWAFisheriesModelling/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 in Microsoft Windows or from a terminal window in Linux. CASAL2 gets its information from input data files, the main one of which is the *input configuration file*. Commands and subcommands in the input configuration file are used to define the model structure, provide observations, define parameters, and define the outputs (reports) for CASAL2. Command line switches tell CASAL2 the run mode and where to direct its output. See Section 3 for details.

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

The *partition* is a representation of the population at an instance in time, and can be considered a matrix of the numbers of individuals within each category and at each age.

A *derived quantity* is a summary of the abundance or biomass in a selected part of the partition at some instance in time. Unlike the partition (which is updated as each new process is applied), a derived quantity records a single value for each year of the model run. Hence, derived quantities build up a vector of values over the time period represented by the model. For example, the total biomass of individuals in categories labelled, say, ‘mature’ at some instance in the annual cycle may be a derived quantity. The derived quantity is then available to the model to be reported, or to be an input into another process (for example, recruitment) at some instance in the model in a subsequent year.

The state at some instance in time is the term for the combination of the partition and any derived quantities at that instance in time. Throughout the model, changes to the state occur from the application of *processes*. This state then provides the basis for the generation of expected values for *observations*, as well as for reports and other outputs.

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

Initialisation can be in one or more phases, and for each phase, the processes that occur in each year, and the order in which they are applied, need to be defined. The processes that occur is controlled by the *annual cycle*. This defines what processes happen in each model year and in what sequence. Further, the processes in each year are 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 how/when observations are evaluated. The user specifies the time-steps, their order, and the processes within each time-step. If more than one process occurs in the same time-step, then they occur in the order that they are specified.

Observations are always linked to a time-step, and are evaluated by the model in 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 manner in which observations are evaluated and how the expected values are calculated by the model is described later in Section 6.

The population structure of CASAL2 follows the usual population modelling conventions and is similar to those implemented in, for example, CASAL (Bull et al., 2012). The model records the numbers of individuals by category and age (e.g., numbers of males and females at age). 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 provides a complete description of the model structure (i.e., population structure, initialisation, and the subsequent 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 that defines the model structure, population dynamics, etc.; the estimation section that defines the methods of estimation (minimisation methods or MDCMC algorithms) and the model parameters to be estimated; the observation section that defines the observational data and associated likelihoods; and the report section that defines the printouts and reports from the model and where these are saved. 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 method and phases, 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 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. Note that the input configuration file can 'include' other files as a part of its syntax. Collectively, these are called the input configuration file.

Other input files can, in some circumstances, be supplied, depending on what is required. For example, a file can be supplied that defines the starting point for estimation, as points from which to simulate observations, or as points from which to run projections.

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 in your systems `PATH`. The CASAL2 installer should update your path on Windows in any case, but 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. 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., Windows10), 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 `config.csl2`.

and where *task* is one of, if there are square brackets **[]** this indicates a secondary label to call the same task for example **`-h`** will execute the same task as **`--help`**;

`-h [--help]` Display help (this page).

`-l [--licence]` Display the reference for the software license (GPL v2).

`-v [--version]` Display the CASAL2 version number.

`-r [--run]` *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 [--estimate]` 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 [--profiling]` 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 [--mcmc]` 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 [--projection]` 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 [--simulation]` *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 [--input] file` *Input* one or more sets of free (estimated) parameter values from `file`. See Section 11 for details about the format of `file`.

`-o [--output]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` 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 3.6.

`-q [--query]` Query an object type to see it's description and parameter definitions. This will print out an extract of the object. An object can be defined as block.type. For example `casal2 --query process.recruitment_constant`, will query the constant recruitment block.

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 NaturalMortality
or
!include MyModelSpecification.csl2
```

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 colon (:), e.g. 1994:1996 is expanded to an integer vector of values 1994 1995 1996),
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.

```
# This is a comment and will be ignored
@process NaturalMortality
m 0.2
/*
This block of code
is a comment and
will be ignored
*/
```

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 `m` of the command `@process` with the label `NaturalMortality` is category related and so, if you want to reference all m related categories you can specify,

```
process[NaturalMortality].m
```

if you wanted to specify teh naturl mortality for single categories you could specify it using,

```
process[NaturalMortality].m{male}
```

3.6. Single stepping CASAL2

Single stepping in CASAL2 gives it the ability to write reports and ‘pause’ after each year in the annual cycle during a run, and then wait and process user input of updated estimable parameters for the next year.

This can allow CASAL2 to be used for implementing models that require feedback management simulations or scenarios, for example for use in operational management procedures (OMPs). This can be automated using **R**, where CASAL2 may be controlled by **R** to update input harvest values (for examples, catches in a fisheries model) to evaluate a particular harvest control rule.

3.7. 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. If configuration errors are found, CASAL2 will print an error messages along with the associated files and line numbers where the errors were identified. For example

```
#1: At line 15 in Reports.csl2: Parameter '{' is not supported
```

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 their 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 population section of a CASAL2 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 group of consecutive mortality based processes, where individuals are removed from the partition. For more information on mortality blocks see Section 4.4 for more detail.

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 how the underlying population changes over time (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 that include recruitment, natural mortality, anthropogenic mortality, ageing, migration, tagging events, and maturation. Different processes may be useful for different models in different circumstances.

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 throughout the year. 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 individuals in the current population into different categories. (Note that the partition records numbers of individuals, not biomass). The individuals are grouped into categories, for example, sex, maturity state, area, and species. However CASAL2 has no predefined categories, and these are defined by the user. This differs from CASAL (Bull et al., 2012) that has only pre-defined partition categories.

The resulting partition can be conceptualised as a matrix, where each row is represented by a category and the columns are the age classes, shown in Figure 4.1. Each row represents the number of individuals for that category in that age class.

The names of categories are user defined, and 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 (but could be defined as 'none'). 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 fish population with a mature and non-spawning fishery. If we assume that the non-spawning fishery happens over most of the year (say 10 months) in the non-spawning 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 non-spawning area. The modeller decides that fish will be divided in the partition by age, sex, maturity, and area (spawning and non-spawning grounds). So the partition has 8 rows (2 sexes (mature or immature) 2 areas) and one column per age class.

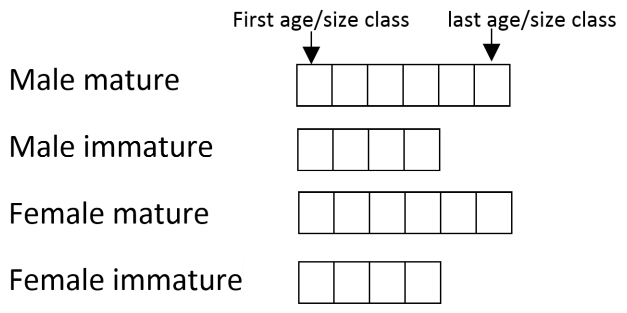

Figure 4.1: A visual representation of a partition

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 non-spawning 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 and where observations occur, but this gives the basic idea.

This structure can be used to implement complex models, with intermingling of separate species and stocks, with complex migration patterns over multiple areas, and multiple sources of anthropogenic impact using different methods and covering different areas and times. However, we note that there is little point in using a complex structure to model a population 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. For information on how to define categories and using CASAL2's shorthand syntax see Section 13.2.

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 matrix that store numbers of individuals at age for each category. A category represents a group of individuals that have the same specific attributes, examples of such attributes include life histories and growth rates, etc. For example, categories may include labels such as:

- Sex (male or female);
- Area (any number of areas, named by the user);
- Maturity (immature or mature);
- Growth-path (any number of growth-paths);
- Tag (any number of tagging events);
- Species

A stock can be thought of as a population of individuals which recruits separately. See Section 4.11 for the treatment of maturity when it is not a category in the partition.

So, you need to tell CASAL2 the following:

- The minimum and maximum age classes in an age-based model.
- Whether there is an age-plus group.
- The names of all categories.

Age classes are always one year wide, except that the maximum age group can optionally be a plus group. Users need to choose the minimum and maximum age classes.

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 in a subset of the model initialisation phases or years (for example, tagging events). Excluding categories for certain years can save a considerable amount of time as CASAL2 does not need to, for example, initialising empty categories or implement processes in time periods when they have no effect.

Another important component of the state object in CASAL2 are derived quantities. This includes quantities such as a mature biomass (for example, in fisheries models, the mid-spawning season biomasses of spawning fish, SSB) for either one or sum of more than one category. CASAL2 derives through the command `@derived_quantity`, and may be required in the specification of some processes (i.e., in fisheries models, a recruitment process that specifies a stock recruitment relationship requires the definition of a derived quantity that specifies the mid-season spawning stock biomass).

4.4. Time sequences

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

- Annual cycle
- 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 time periods and processes. Any number of processes can occur within each time-step, in any order (although there are limitations around mortality based processes - see Section 4.4) 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, optionally, be different from that which is applied during the model years.

Mortality blocks

For every time step in an annual cycle there is an associated *mortality block*. Mortality blocks are a key concept in CASAL2.

Mortality blocks are used to define the 'point' in the model time sequence when observations (see Section 6) are evaluated, and derived quantities (see Section 4.8) are calculated.

A mortality block is defined as a consecutive sequence of mortality processes within a time step. The processes that are mortality processes are all pre-defined in CASAL2, and cannot be modified. These include all the processes described in subsection 4.7.3.

CASAL2 requires that each time step has exactly one mortality block. The achieve this, either all the mortality processes in a time step must be sequential (i.e., there can not be a non-mortality process between any two mortality processes within any one time step); or if no mortality processes occur in a time step then the mortality block is defined to occur at the end of the time step.

CASAL2 will error out if more than one mortality block occurs in a single time step.

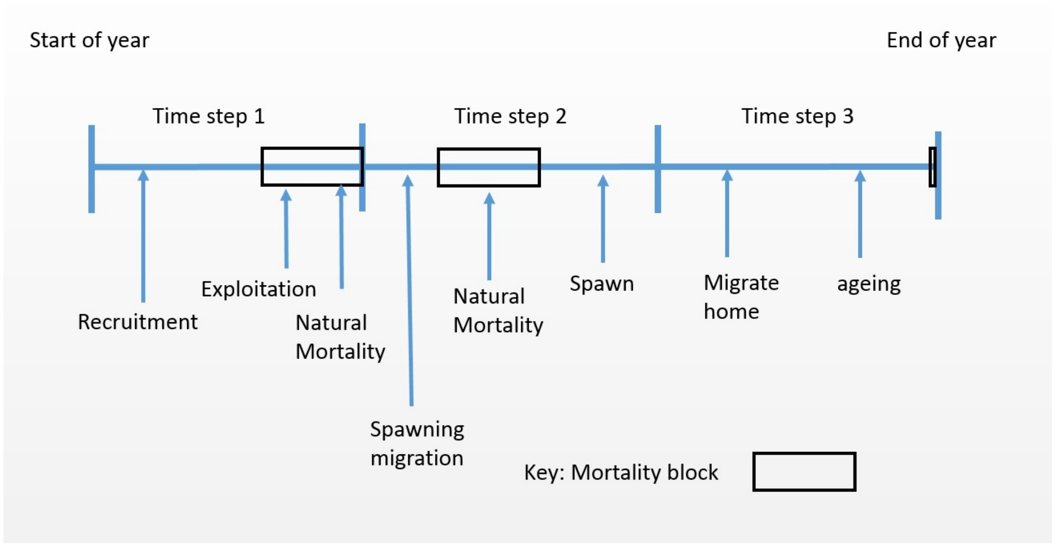

Figure 4.2: A visual representation of a hypothetical sequence for an annual cycle.

4.4.1. Initialisation

Initialisation is the process of determining the model starting state, whether it be equilibrium/steady state or some other initial state for the model (e.g exploited), prior to the start year of the model. This can be computationally expensive if a plus group is present in the partition.

There are multiple methods to initialise a partition in CASAL2. These methods are: iterative, fixed, derived, and Cinitial.

Model initialisation can also occur in several phases, each of which can be a different method. These are carried out in sequence. At the end of all of the initialisations, CASAL2 then runs the model years carrying out processes in each time step in the annual cycle.

The multi-phased initialisation allows the user to choose a number of initialisations that may assist with optimising the models for speed, initialise a non-equilibrium starting state, or resolve simple processes before introducing more complex ones.

Each phase of the initialisation defaults to have the same processes and in the same order as defined in the annual cycle. Although they can involve any number of processes using the `insert_processes` subcommand.

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.

Note that the *first* initialisation phase is always initialised with each element (i.e., each age and category) set at zero. Note that you may need to be careful when using complex category inter-relationships or density dependent processes that depend on a previously calculated state, as they may fail when used in the first phase of an initialisation.

Multi-phase iterations can also be used to determine if the initialisation has converged. Here, add a second initialisation phase for, say, 1 year (with the same processes applied). Then report the state at the end of the first and second phase. If these states are identical, then its likely that the initialisation has converged to an equilibrium state.

Syntax for including or excluding processes through the `insert_processes` and `exclude_processes`. For the `insert_processes` the syntax is;

```
insert_processes time_step_label(process_label_in_annual_cycle) = label_new_process
```

An example of this is would be in the `@time_step` labelled `Oct_Nov` include the `@process` labelled `predationIni` before the `@process` labelled `Instantaneous_Mortality`

```
insert_processes Oct_Nov(Instantaneous_Mortality)=predationIni
```

if you want to include a process at the end of the time step you can use the following syntax,

```
insert_processes Oct_Nov()=predationIni
```

To exclude a process from an initialisation phase the syntax is much simpler and can be done by the following subcommand in your `@initialisation_phase`;

```
exclude_processes process_label_in_annual_cycle
     exclude_processes Instantaneous_Mortality
```

The command above will remove the process `Instantaneous_Mortality` during that particular initialisation phase.

Iterative Initialisation

The iterative initialisation is a general solution for initialising the model. The iterative method can be slow to converge, depending on the nature of the problem being resolved, but will work on even complex structured models that may be difficult or impossible to implement using analytic approximations.

The number of iterations in the iterative initialisation can effect the model output, and these should be chosen to be large enough to allow the population state to fully converge. We recommend that a period of about two generations to ensure convergence. CASAL2 can be requested to report a number of convergence statistics that can assist the user determine the level of convergence.

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

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

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).

An example of the syntax to implement this would be,

```
@model
...
initialisation_phases Iterative_initialisation

@initialisation_phase Iterative_initialisation
type iterative
years 50
lambda 0.0001
convergence_years 20 40
```

Derived Initialisation

`Derived` initialisation is an analytical solution that calculates the equilibrium age structure and the plus group using a geometric series solution. 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 (for example. one way migrations) that this solution does not reach the exact equilibrium partition. We note that if using this method, that users confirm the partition has reached an equilibrium state by either comparing with and iterative initialisation, or by adding a second iterative initialisation phase of a limited number of iterations to confirm convergence.

An example of the syntax to implement this would be,

```
@model
...
initialisation_phases Equilibrium_initialisation

@initialisation_phase Equilibrium_initialisation
type derived
```

Cinitial Initialisation

This initialisation is only available as a second or greater phase initialisation, and can only be applied after derived or iterative initialisation phases. The 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. Note that it may be advisable to include an observation of age composition data for the first year of the model in order to estimate the non equilibrium population state.

An example of the syntax to implement this would be,

```
@model
...
initialisation_phases Cinitial

@initialisation_phase Cinitial
type cinitial
categories spawn.male+nonspawn.male  spawn.female+nonspawn.female
table n
spawn.male+nonspawn.male   5e7 5e7 7e6 6e6 5e6 4e6 3e6 2e6 1e6 1e6 1e1 1e1 1e1 1e1
spawn.female+nonspawn.female 5e7 5e7 7e6 6e6 5e6 4e6 3e6 2e6 1e6 1e6 1e1 1e1 1e1 1e1
end_table
```

The cinitial factors can also be estimated using the following syntax

```
@estimate cinit_male
parameter initialisation_phase[Cinitial].spawn.male+nonspawn.male
same initialisation_phase[Cinitial].spawn.female+nonspawn.female
lower_bound  2e2  2e2  2e2  2e2  2e2  2e2  2e2  2e2  2e2  2e2  2e0  2e0  2e0  2e0
upper_bound  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9  2e9
type uniform
```

Fixed Initialisation

This is a user defined table that is taken to be the initial partition prior to the start year. Users have the ability to initialise models by specify the numbers at age for each category. Be careful when initialising models with this type, if the model applies processes that require derived quantities to be calculated in the initialisation phase. As this may cause undefined behaviour.

An example of the syntax to implement this would be,

```
@model
...
initialisation_phases Fixed

@initialisation_phase Fixed
type state_category_by_age
categories male female
min_age 3
max_age 10
table n
male   1000 900 800 700 600 500 400 700
female 1000 900 800 700 600 500 400 700
end_table
```

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, within each time-step.

Processes are carried out in the order specified within each time-step. These can be the same or different to the processes in 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

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. Users invoke CASAL2 to run in projection mode using the following command `casal2 -f 1`. The number the follows the `-f` parameter indicates how many projections you would like to undertake for each set of parameters supplied. This allows you to explore many future scenarios for a single set of parameters, this value should only be greater than 1 if you are applying a projection type that is stochastic. When running projections after a bayesian analysis you should add the `--tabular` parameter into the command. This will output a tabular report (see Section 7.19) which is easier to manipulate and run diagnostics on in the **R** environment.

Projection years occur immediately after the model run years and defined as the `final_year + 1` upto and including `final_projection_year`.

For a projection run in CASAL2 a model is initialised and run through the model years from `initial` to the `final`. During this phase CASAL2 stores all parameter values so that projection classes can (if the users choices) allow parameters before `final_year` to be projected (See end of this paragraph for an example of when you may want to do this). The model then is re-run from `initial` to `projection_final_year`, where any parameter can be either fixed or, if specified, drawn from a stochastic distribution or process during that time period. An example of when users may want to begin projecting a parameter before the projection phase has started is for year class parameters. Usually the last few year class parameters are poorly estimated (if they are estimated at all), although this depends on the quality and coverage of the compositional data that would inform these parameter or the precenese of a recruitment index. So users may wish to assume they are unknown and apply projection methods as they do for future values.

CASAL2 does not have any default projections for when parameters are specified by year. These must be specified using the `@project` command blocks. This is important for parameters that may vary from year to year (such as year class strength parameters), CASAL2 should error out if users run CASAL2 in projection mode but have not specified a `@prject` block for the `ycs_values` parameter.

CASAL2 allows any estimable parameter to be specified in a `@project` block and then used in a projection. The available projection types for these parameters include constant, lognormal, empirical-lognormal, empirical re-sampling, or user-defined. To see all the projection classes available and examples of their syntax see Section 4.13.

The subcommand that is common to all projection methods is the `years`,`parameter` and `multiplier` command. This is as it suggests a constant that is multiplied to the projected value after it has been derived by its respective method.

A **note** to users for the specific year class parameter, The definition of year applies to the `ycs_years` not the model years. As defined in Section 4.7.1, `ycs_years` are offset between time of spawning and when they enter the partition.

4.7. Population processes

Population processes are those processes that change the model state. Processes produce changes in the model partition, by adding, removing or moving individuals between ages and/or categories. The population processes include recruitment, ageing, mortality events (e.g., natural and anthropogenic) 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 `mortality_constant_rate` and `mortality_instantaneous`. There are also processes that only occur within the time step they are defined. Each of these processes are 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 partition. CASAL2 currently 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, or multiple recruitment processes can be defined. For example, if recruiting to categories labelled male and female, then you might set the proportions as 0.5 and 0.5 respectively to denote that half of the recruits recruit to the male category and the remaining half to the female category.

An important note here is that recruitment can differ from a spawning event or the creation of a cohort/year class. In a fisheries context recruitment usually refers to indivisuals 'recruiting' to the fishery. This is done for a few reasons, one being often we do not have a lot of information relating to age classes between spawning and data collection i.e. an information gap exists. Once again in a fisheries context this information gap can refer to the time between spawning and being vulnerable to a survey or fishery for data collection. Thus users may only want to model the population for which data is available. This offset between spawning and recruitment is parameratised either by the recruitment variable `age` or `min_age` (the default value for the `age` parameter in the recruitment process). For CASAL users the parameter `age` is the same as `y_enter` in CASAL.

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

$$N_{y,a,j} \leftarrow N_{y,a-1,j} + p_j(R_{y,a}) \tag{4.2}$$

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

Constant Recruitment

In the constant recruitment process the total number of recruits added each year y in age a is $R_{y,a}$, and is simply $p_j(R_0)$, i.e.

$$R_{y,a,j} = p_j(R_0) \tag{4.3}$$

Constant recruitment recruits a constant number of individuals each year. It is equivalent to a Beverton-Holt recruitment process with steepness set equal to one (i.e., $h = 1$).

For example, to specify a constant recruitment process, where individuals are added to male and female immature categories at $age = 1$ evenly (`proportions` = 0.5), and the number to add is $R_0 = 5 \times 10^5$, then the syntax is

```
@process Recruitment
type constant_recruitment
categories male.immature female.immature
proportions 0.5 0.5
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.,

$$R_{y,a,j} = p_j(R_0 \times YCS_{ycs\_year} \times SR(SSB_{ycs\_year})) \tag{4.4}$$

where

$$ycs\_year = y - \texttt{ssb\_offset} \tag{4.5}$$

where a is the parameter `age`, p_j is the proportion of recruits to enter category j and `ssb_offset` is the lag between spawning and recruitment. As stated earlier recruitment refers to the recruitment into the population and may differ from the spawning event. See two paragraphs below on more information about `ssb_offset`, but in general this parameter shouldn't be specified by the user.

SR is the Beverton-Holt stock-recruit relationship parametrised by the steepness h,

$$SR(SSB_y) = \frac{SSB_y}{B_0} / \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB_y}{B_0}\right)\right) \tag{4.6}$$

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, this initialisation phase is defined by the parameter `b0_intialisation_phase`. 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 in the annual cycle, for this reason users can choose to initialise models using B_0 or R_0.

`ssb_offset` should not be manually given by the user in commonplace, CASAL2 determines `ssb_offset` by the order of ageing, recruitment, spawning and the recruitment parameter `age`,

- If recruitment then ageing then spawning, then `ssb_offset` should equal `age` + 1.
- If spawning then ageing then recruitment, then `ssb_offset` should equal `age` - 1.
- else `ssb_offset` = `age`

The certain scenarios where the user will manually want to input these values is if there are multiple ageing processes in the annual cycle. We have not coded CASAL2 to deal with this situation so will be up to the user to define the `ssb_offset`.

Another important input parameter is the `ycs_years` which is defined in Equation (4.5). When referencing the parameter `ycs_values` you should always reference it by the `ycs_years` parameter, this is important to note for when defining `@estimate`, `@project` and `@time_varying` blocks for the parameter `ycs_values`. An example follows

Year classes values are standardised using the Haist parametrisation suggested by V. Haist. Here, the model parameter `ycs_values` is a vector **Y**, covering years from `start_year - ssb_offset` to `final_year - ssb_offset`, as defined by the parameter `ycs_years`. The year class strengths are calculated by $YCS_i = Y_i/\bar{\mathbf{Y}}$ where the mean is calculated over the user-specified years `standardise_ycs_years`. Then,

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

where S is the set of years from `standardise_ycs_years`. One effect of this parametrisation is thatR_0 is then defined as the mean estimated recruitment over the years S, because the mean year class multiplier over these years will always be one.

Typically, the user will define `standardise_ycs_years` to span the years for which they expect to have reasonable estimates of YCSs. Often, the user will wish to force $Y_y = 1$ for some or all years $y \in S$ (this is equivalent to forcing Ry=R0 x SR) by setting the lower and upper bounds of these Ys to be 1. An exception to this might occur for the most recent YCSs, which the user may want to estimate, but not include in the definition of R0 (because the estimates are based on too few data). Note that, optionally, the user may exclude one or more years from within the range from the averaging process of the Haist parameterisation.

The advantage of the Haist parameterisation is that the user need no longer use a large penalty to force the mean of the YCS parameter to be 1 (though they should still use a small penalty to stop the mean of Y from drifting). This may improve MCMC performance. projected YCS are not affected by this feature. A disadvantage with this parameterisation in a Bayesian analysis is that the prior refers to the Ys, not the YCS.

An example of a the configuration of the Beverton-Holt recruitment process, where individuals are added to the category ‘immature’ at $age = 1$, and the number to add is $R_0 = 5 \times 10^5$. `SSB_derived_quantity` is a derived quantity that specifies the total spawning stock biomass that contributed to the this year class, 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 1995 to 2004, and recruits enter into the model two years following spawning.

```
@process Recruitment
type recruitment_beverton_holt
categories immature
proportions 1.0
r0 500000
b0_initialisation_phase phase1
steepness 0.75
age 1
ssb SSB_derived_quantity
standardise_ycs_years 1995:2004
ycs_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
ycs_values   0.65    0.87    1.6    1.13    1.0235    0.385   2.653    1.35    1    1    1    1    1
```

So, to specify a Beverton-Holt recruitment for each stock, the following information is required:

1. YCS, starting from year (`start_year` - `ssb_offset`) and extending up to year (`final_year` - `ssb_offset`).
2. The value of `age` aka `y_enter` to CASAL users.
3. the steepness parameter `h`.
4. In a multi category model, the proportion of recruits for each category.
5. label for the derived quantity

Defining a `@recruitment` block with a `@initialisation_phase` (Section 4.4.1) type = `derived`. Ususally in a recruitment processes you only need to define the categories that recieve recruits, for example if you had a model that had a spawning area different from the are that recruits enter the population at, you would want an area explicit model containing spawning categories and recruiting categories. You can just specify the recruited categories in the subcommand `categories`. As these will be the categories recieving recruits. **If** you have `@initialisation_phase`, `type=derived` you need to specify all categories that are apart of that recruitment process. using the following example this would be defined as;

```
@process Recruitment
type recruitment_beverton_holt
categories recruits.male recruits.female spawn.male spawn.female
proportions 0.5 0.5 0.0 0.0
r0 500000
ssb SSB
....
```

note the proportions = 0.0, this is needed because of how the derived initalisation phase works, it finds a solution for when `r0` = 1.0 based on a infinite geometric series for the plus group, and scales the initial partition by `r0`. Thus if you did not include all categories not all categories would be initialised to true values and give possibly messed up answers. This is cauton and will come up again in fishery based models where you have multiple stocks then you need to list all the categories that make up the stock.

4.7.2. Ageing

The ageing process 'ages' individuals — it simply moves all individuals in the named categories i to the next age class $j+1$, or accumulates them if the last age class is a plus group.

The ageing process is defined as,

$$\text{element}(i,j) \leftarrow \text{element}(i,j-1) \tag{4.7}$$

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

$$\text{element}(i,age_{\text{max}}) \leftarrow \text{element}(i,age_{\text{max}}) + \text{element}(i,age_{\text{max}-1}). \tag{4.8}$$

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

Six types of mortality processes are permissible in CASAL2, constant rate, event, biomass-event, Hollings mortality, initialisation mortality 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. Note that CASAL2 does not (yet) implement the Baranov catch equation but instantaneous mortality is an approximation to the baranov catch equation. To apply both natural and biomass-event mortality, users can use `mortality_instantaneous`. Note that all mortality processes occur within a mortality block of a time step see Section 4.4 for more information and definitions on mortality blocks.

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
```

$$D_{j,t} = \sum_{a} N_{a,j}(1 - \exp S_{a,j} M_j p_t) \tag{4.9}$$

Where, $D_{j,t}$ is the number of deaths in category j in time step t, $N_{a,j}$ is the number of individuals in category j at age a. $S_{a,j}$ is the selectivity value for age a in category j, M_j is the mortality rate for category j, and p_t is the proportion of the mortality rate to apply in time step t.

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. These can be used to include anthropogenic mortality where numbers of removals are known, for example, fishing in a fisheries model, rather than applying mortality as a rate.

In these cases, the abundance or biomass removed is also 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. When minimising, event mortality processes require a penalty 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 too low a number of individuals in the defined categories (after applying selectivities) to allow for removals at the maximum exploitation rate. See Section 5.8 for more information on how to specify 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 $1\dots I$ for ages $j = 1\dots J$ that are subject to event mortality,

$$V(i,j) = S(j)N(i,j) \tag{4.10}$$

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

$$V_{total} = \sum_i \sum_j V(i,j) \tag{4.11}$$

Hence the exploitation rate to apply is

$$U = \begin{cases} C/V_{total}, & \text{if } C/V_{total} \le U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.12}$$

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

$$R(i,j) = UV(i,j) \tag{4.13}$$

For example, to specify an **abundance based** fishing mortality process in a fisheries model, with catches given for a set of specific years, over categories ‘immature’ and ’mature’, with selectivity ‘FishingSel’ and assuming a maximum possible exploitation rate of 0.7, then the syntax would be,

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

Another example of a **biomass** based fishing mortality process with the same characteristics as earlier.

```
@process Fishing
type mortality_event_biomass
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 process that combines both natural mortality and event biomass mortality into a single process. This allows the natural mortality to occur occurs across multiple time steps, and can specify multiple instances of event mortality to account for, say, multiple fisheries operating sequentially or concurrently. This process applies half the natural mortality in each time step, then the mortalities from all the concurrent fisheries instantaneously, then 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., $U_{f,obs} = U_f$), but they can be different 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 $U_{f,max}$ for each fishery f. So, no more than proportion $U_{f,max}$ 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 U_max.

  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 0.20 across three time steps on both male and female categories. And we have two method of removals (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 0.20
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 method
```

```
method       category  selectivity u_max   time_step penalty
FishingWest   stock     westFSel    0.7     step1     CatchPenalty
FishingEast   stock     eastFSel    0.7     step1     CatchPenalty
end_table
```

Referencing catch parameters for use in projecting, time_varying and estimating uses the following syntax;

```
parameter process[mortality_instantaneous].method_"method_label"{2018}
```

where `"method_label"` is lower case from your `catch` or `method` table taking the following earlier example;

```
parameter process[instant_mort].method_FishingWest{2018}
```

4.7.3.4. Hollings mortality rate

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

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

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

$$P(k,l) = S_{predator}(l) N_{predator}(k,l) \tag{4.14}$$

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

$$P_{total} = \sum_K \sum_L P(k,l) \tag{4.15}$$

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

$$V(k,l) = S_{prey}(l) N_{prey}(k,l) \tag{4.16}$$

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

$$V_{total} = \sum_K \sum_L V(k,l) \tag{4.17}$$

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

$$R_{total} = P_{total} \frac{a V_{total}^{x-1}}{b + V_{total}^{x-1}} \tag{4.18}$$

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

Hence the exploitation rate to apply is

$$U = \begin{cases} R_{total}/V_{total}, & \text{if } R_{total}/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.19}$$

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

$$R(k,l) = UV(k,l) \tag{4.20}$$

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

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

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

4.7.3.5. Initialisation event or biomass-event mortality

The Initialisation event mortality process are specific processes that only can occur in the initialisation phase. It allows users to apply abundance or biomass mortality events specifically in initialisation phases. This can be useful if you wanted to deviate a model from equilibrium before model start. This process applies a single catch for all iterations within the initialisation phase, this process wont apply any mortality outside of the initialisation phase. It is advised that users use this process in conjunction with the `insert_processes` command in the `@initialisation_phase` block, and not embed this process in the annual cycle. Example syntax to implement such a scenario,

```
initialisation_phases Equilibrium_state Predation_state
time_steps Oct_Nov Dec_Mar

@initialisation_phase Equilibrium_state
type derived

@initialisation_phase Predation_state
type iterative
insert_processes Oct_Nov()=predation_Initialisation

@process predation_Initialisation
type initialisation_mortality_event
categories male.HOKI female.HOKI
catch 90000
selectivities Hakesl Hakesl
```

```
time_step Oct_Nov
processes Mg1 Instantaneous_Mortality

@time_step Dec_Mar
processes Recruitment Instantaneous_Mortality
```

Note how we have added the `initialisation_mortality_event` into the initialisation phase `Predation_state` but not in the annual cycle. This was a case where the functionality has been applied.

4.7.4. Transition By Category

This process covers moves individuals between categories. Because the CASAL2 partition user defined, this type of process is used to move individuals between categorised, and is used to specify processes such as maturation (move individuals from an immature to mature state) or migration (move individuals from one area to another).

4.7.4.1. Annual transition by category

A special case is annual transition by category, which allows a transition to occur in a specific subset of years only, where each year can have a different rate.

In both cases, there has to be a one to one relationship between the 'from' category and the 'to' category — for every source category there is one target category. If however, you want to merge categories, then just repeat the 'to' category multiple times.

$$N_{a,j} = N_{a,i} \times P_i \times S_{a,i} \tag{4.21}$$

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 transition_category
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 the known lengths of release. CASAL2 also allows for initial tag loss. An example of tag release by length process is as follows.

```
@process 2005Tags_shelf
type tag_by_length
years 2005
from male.untagged  female.untagged
to male.2005  female.2005
selectivities MatureSel
proportions 1
selectivities ShelfselMale ShelfselFemale
penalty tagging_penalty
maximum_length 230
plus_group False
initial_mortality 0.1
table proportions
year 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220
2005  0 0 0.0580 0.1546 0.3380 0.1981 0.1643 0.0531 0.0242 0.0097 0 0 0 0 0 0 0 0 0 0
end_table
n 207
U_max 0.999
```

The above process will move 207 individuals from a combination of male.untagged and female.untagged, based on the combination of growth rates and selectivity between the sexes.

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 migration rate that can happen over multiple time steps in the annual cycle. This method assumes when tags are lost that the fish are transferred from the `from` category to the `to` category. How the tag loss rate is applied depends on whether the fish were only tagged with a single tag (`tag_number_per_animal = 1`), double tagged (`tag_number_per_animal = 2`) or *n* tagged (`tag_number_per_animal = n`). The relationship is as follows

```
@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 at 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 4.4 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 $n_{p,j} = n_j^{1-p} n_j'^p$

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,

$$\bar{s}(age) = L_\infty \left(1 - \exp\left(-k\left(age - t_0\right)\right)\right) \tag{4.22}$$

Schnute: where length at age is defined as,

$$\bar{s}(age) = \begin{cases} \left[y_1^b + (y_2^b - y_1^b)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right]^{1/b}, & \text{if } a \neq 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right], & \text{if } a \neq 0 \text{ and } b = 0 \\ \left[y_1^b + \left(y_2^b - y_1^b\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right]^{1/b}, & \text{if } a = 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right], & \text{if } a = 0 \text{ and } b = 0 \end{cases} \tag{4.23}$$

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

Length-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

$$\text{mean weight} = 1 \tag{4.24}$$

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

$$w = al^b \tag{4.25}$$

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

$$w = (al^b)(1+cv^2)^{\frac{b(b-1)}{2}} \tag{4.26}$$

  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).

4.10. Weightless model

If users would like to only model abundance and are not interested in converting the population to weight you can turn off the `@length_weight`, by specifing the keyword `none` in the `@age_length` block. For example,

```
@age_length age_size
type schnute
...
length_weight none
```

If this specified in the model any biomass that is generated by CASAL2 will actually be abundance, so be careful on intepretations when using this setting.

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 an 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

$$f(x) = C \tag{4.27}$$

The constant selectivity has the estimable parameter C.

4.12.2. `knife_edge`

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{4.28}$$

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`

$$f(x) = V_x \tag{4.29}$$

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

4.12.4. `all_values_bounded`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \tag{4.30}$$

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

4.12.5. `increasing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \le x \le H \\ f(\alpha), & \text{if } x \ge H \end{cases} \tag{4.31}$$

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`

$$f(x) = \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.32}$$

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`

$$f(x) = \alpha - \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.33}$$

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`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1))\,/\,(1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \ge H \end{cases} \tag{4.34}$$

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`

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \le \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \ge \mu \end{cases} \tag{4.35}$$

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 $x = a_1 + s_R$.

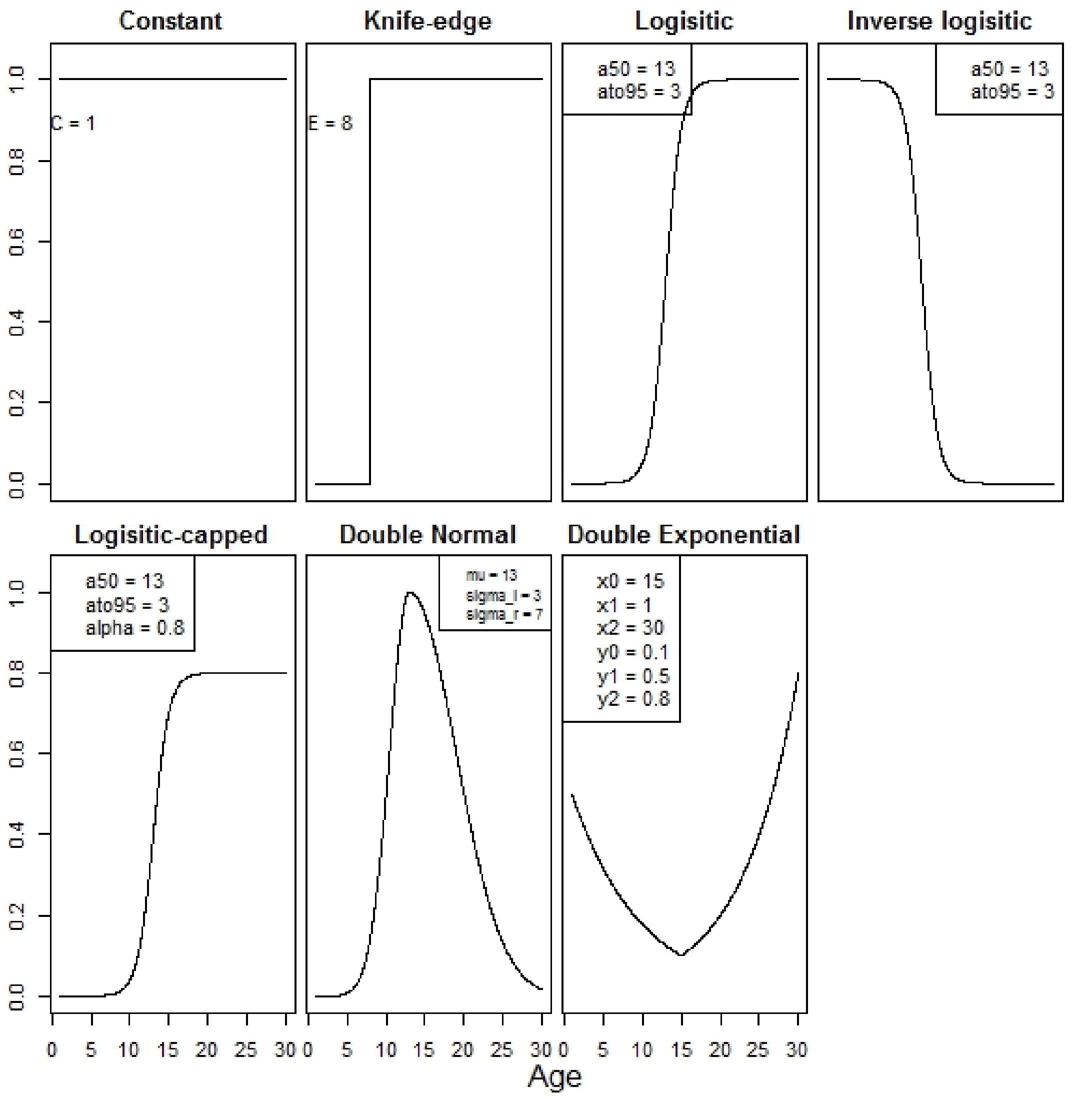

Figure 4.3: Examples of the functional forms of selectivities available in CASAL2.

4.12.10. `double_exponential`

$$f(x) = \begin{cases} \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \le x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{4.36}$$

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).

Selectivities `all_values` and `all_values_bounded` can be addressed in additional priors using the following syntax,

```
@selectivity maturity
type all_values
```

```
v 0.001 0.1 0.2 0.3 0.4 0.3 0.2 0.1

## encourage ages 3-8 to be smooth.
@additional_prior smooth_maturity
type vector_smooth
parameter selectivity[maturity].values{3:8}
```

4.13. Projections

This section lists all the projections classes available, their functionality and an example of the syntax.

Constant

A parameter can either be fixed during all projection years or specified individually for each projection year. This is a deterministic assumption, where the parameter is assumed to be known without error during projection years.

```
@project Future_ycs
type constant
parameter process[Recruitment].ycs_values
years 2012:2016
values 1 2 1 2 0.5
multiplier 1
```

Empirical resampling

Parameters that have time components associated with them can be re-sampled uniformly with replacement over a range of years and used as the projected year values. The year range which users must specify are between `start_year` and `final_year`

```
@project Future_ycs
type empirical_sampling
parameter process[Recruitment].ycs_values
years 2012:2016
start_year 1988
final_year 2008
multiplier 1
```

Lognormal

The parameters are originally drawn from a gaussian distribution in log space and exponentiated out to form the lognormal distribution,

$$X_p = e^{\varepsilon_p - 0.5\sigma^2} \tag{4.37}$$

where $\varepsilon_p \overset{iid}{\sim} N(\mu, \sigma)$ and X_p is the projected value for parameter X, and μ, σ is the mean and standard deviation on the log scale. An example of applying this process is if we wanted to draw future year class parameters from a lognormal distribution with mean 1 and standard deviation 0.8, we would define the syntax as,

```
@project Future_ycs
type lognormal
parameter process[Recruitment].ycs_values
years 2012:2016
mean 0
sigma 0.8
multiplier 1
```

Lognormal-Empirical

This class applies Lognormal draw as in the `LogNormal` class but it allows the user to specify a year range which is re-sampled uniformly without replacement. These re sampled values are then used to calculate the standard deviation of the distribution. Then equation (4.37) is used to generate future values with user defined μ and empirically calculated σ,

```
@project Future_ycs
type lognormal_empirical
parameter process[Recruitment].ycs_values
years 2012:2016
mean 0
start_year 1988
final_year 2008
multiplier 1
```

User Defined

This class allows users to use the equation parser to define the future values of a parameter during projection mode. This was originally set-up to apply harvest control rules (i.e. apply a management action, change in TACC based on the current or previos state). In fisheries this would be setting the catch based on an exploitation rate * vulnerable biomass, where the exploitation rate is based on some rule as shown in figure **??**.

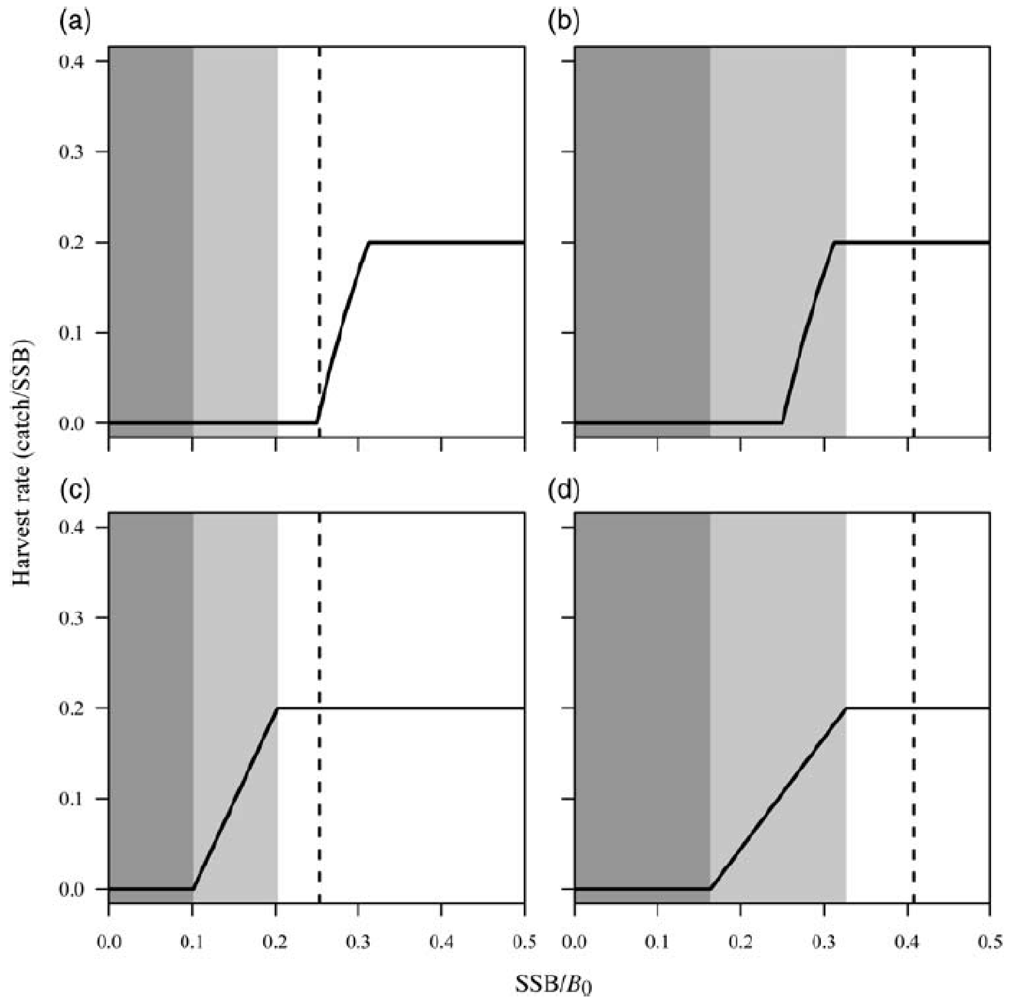

Figure 4.4: Examples of control rules based on current stock status.

```
@project HCR_2015
type user_defined
parameter process[Instantaneous_Mortality].method_Sub_Ant_F
years 2015
equation if(derived_quantity[SSB].values{2014} / process[Recruitment].b0 <= 0.1, 0.0,
if(derived_quantity[SSB].values{2014} / process[Recruitment].b0 > 0.1 &&
derived_quantity[SSB].values{2014} / process[Recruitment].b0 < 0.2,
derived_quantity[SSB].values{2014} * derived_quantity[SSB].values{2014}
/ process[Recruitment].b0,
derived_quantity[SSB].values{2014} * 0.2))
```

The syntax of the equation parser is a bit difficult and care should be taken when writing it. In words the above equation is, if $\%B_{2014} \leq 0.1$ next years catch is 0.0, else if $\%B_{2014} > 0.1 \& \%B_{2014} \leq 0.2$, next years catch is equal to $\%B_{2014} * SSB_{2014}$, else set next years catch to $0.2SSB_{2014}$

4.14. Time Varying Parameters

CASAL2 has the functionality to vary any 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. Method `type`'s for time varying a parameter are; `constant`, `random_walk`, `exogenous`, `linear`, `annual_shift`, `random_draw`. This allows users to let a parameter be known in a year, be the result of a deterministic equation or stochastic. **Note** that we haven't implemented random effects so be careful on your intepretation if you choose to estimate parameters that control any stochastic time-varying method. This functionality was originally implemented for simulating more realistic observations. When you allow removals to have annual varying catchabilities, selectivities and other more realistic model components simulated observations become more real data and thus conclusions based on simulated data are more useful. The other driver was allowing mean or location parameters of selectivities to change between years based on an explanatory variable. An example of this is in the New Zealand Hoki fishery where we allow the μ and a_{50} parameters to shift depending on when the fishing season. Discriptive analysis showed that when fishing was earlier relative to other years they caught smaller fish and vice versa. This can be shown in the Examples 2stock directory, implemented at line: 382 in population.csl2.

4.14.1. `Constant`

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

```
@time_varying q_time_var
type constant
parameter catchability[survey_q].q
years 1975:1988
values 0.001
```

Users can also estimate these year values using the following syntax, a word of caution is that you shouldn't estimate the actual parameter and its time varying counterpart, as the time varying value will overwrite the actual parameter making the first value unidentifiable and an impossible optimization process.

```
@estimate q_time_var
type uniform
parameter time_varying[q_time_var].values(1975:1976)
lower_bound 1e-6 1e-6
upper_bound 2 2
```

4.14.2. `Random Walk`

A random deviate added into the last value drawn from a standard normal distribution. This has an estimable parameter σ_p for each time varying parameter p. 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 almost impossible.

```
@time_varying q_time_var
type random_walk
parameter catchability[survey_q].q
distribution normal
mean 0
sigma 3
```

If the `parameter` specified in the `@time_varying` is associated with an `@estimate` block then the parameter is constrained to stay within the lower and upper bounds of the `@estimate` block.

WARNING, if the parameter does not have an associated `@estimate` block then there is no safe guard for a random deviate to put the parameter in a space where the model fails, i.e generates NA or INF values. to avoid this from happening it is recommended you specify an `@estimate` block even though you are not estimating the parameter like below. A constraint whilst using this functionality is that a parameter cannot be less than 0.0, if it is CASAL2 sets it equal to 0.01.

```
@estimate survey_q_est
type uniform
parameter catchability[survey_q].q
lower_bound 1e-6
upper_bound 10
```

This will insure the random walk time varying process will set the any new candidate within the lower and upper bound of the `@estimate` block.

4.14.3. `Annual shift`

A parameter generated in year y (θ'_y) depends on the value specified by the user (θ_y) along with three coefficients a, b and c as follows,

$$\bar{\theta}_y = \frac{\sum_y^Y \theta_y}{Y} \tag{4.38}$$

$$\theta'_y = a\bar{\theta}_y + b\bar{\theta}_y^2 + c\bar{\theta}_y^3 \tag{4.39}$$

4.14.4. `Exogenous`

Parameters are shifted based on an exogenous variable, an example of this is an exploitation selectivity parameters that may vary between years based on known changes in exploitation behaviour such as season, start time, and average depth of exploitation.

$$\delta_y = a(E_y - \bar{E}) \tag{4.40}$$

$$\theta'_y = \theta_y + \delta_y \tag{4.41}$$

where δ_y is the shift or deviation in parameter θ_y in year y to generate the new parameter value in year y (θ'_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).

4.15. Equation Parser

CASAL2 has the ability to use an equation parser, this is currently implemented in two classes of the model, Projections (section 4.13), Derived quantities (section 4.8) and reports (section 7). Examples of syntax for implementing the equation parser follow from here. But for a more detailed look at the parser see `https://github.com/nickgammon/parser/blob/master/parser.cpp`

```
equation process[Recruitment].r0 * (2-1)
```

Can apply routine mathematical functions such as `log`, `exp`,`cos`,`sin`,`tan` such as;

```
equation sqrt(process[Recruitment].r0)
```

Apply exponents,

```
equation pow(2, 3)
```

evaluate the absolute of an equation using the `abs()` syntax,

```
equation abs(sqrt(process[Recruitment].r0) * 1.33)
```

Use if else statements

```
equation if(process[Recruitment].r0 > 23, 44, 55)
## if R0 is greater than 23 return 44 else return 55
```

You can link if else statements as well but be warned the syntax becomes a bit messy, such as

```
equation if(process[Recruitment].r0 > 23, 44,
          if(process[Recruitment].r0 > 10, 55, 66))
## if R0 is greater than 23 return 44 else if R0 less than 23 but greater than 10 return 55,
else R0 must be less than 10 return 66
```

Constraints, are that you can only reference singletons so cannot apply an equation to vector parameters for example, you can't reference `process[Recruit].ycs_values{1974:1980}`. To see which parameters you can include in an equation parser go to the syntax section (Section 12) any subcommand that has a **Type** Estimable should in theory be addressed in an equation parser.

With the equation parser it is difficult to catch all user configuration errors. Such as we cannot check whether a parameter that exists in the system has been populated when the user requires it. For example if you wanted to say that next years (2015) removals was based on the this years (2014) state of the population. The user can easily specify the wrong year;

```
parameter process[removals].catch
year 2015
equation derived_quantity[percent_b0].values{2020}
```

The above would be an acceptable equation but obviously will cause non-sensical results, because you are asking for a value in 2020 when you are in 2015. This is just a caution that although the equation parser is great and adds a great deal of flexibility, users should be careful because it is easy to mis-specify and near impossible for the development team to catch all errors.

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 **??**. In addition to likelihoods, priors (see Section 5.7) and penalties (see Section 5.8) are components of the objective function. Note that if the priors are specified as uniform, then the prior contribution is zero and the estimation problem turns into penalised-likelihood and not Bayesian.

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. Equation 5.1 can mathematically reduce to a penalised likelihood equation if all priors are assumed to be uniform. This is because uniform priors have a zero contribution to the objective function so Equation 5.1 reduces to likelihoods plus penalties.

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 male.m
parameter process[NaturalMortality].m{male}
lower_bound 0.1
upper_bound 0.4
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 `-e`, profile `-p`, or MCMC `-m` 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. Some minimisers convert lower and upper bound into a minimisation space (for example -1,1 space for the numerical differences algorithm). 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 (`de_solver`), and the dlib minimiser. There are also three auto differential (AD) minimisers being: ADOL-C, CPPAD, and BETADIFF. For references see section 1.5. AD minimisers are recommended for complex models they are on average much faster and from my experience find a better minimum if exploring a complex objective surface.

Recently with the number of parameters growing in these models, an important input parameter on most minimisers is the `tolerance` parameter. This is the stopping rule that minimisers use to define when they have found a 'solution' (remember there is no guarantee that a solution is the global solution, such is the world we live in). Try alternative starting values, this is easiliy done with the `-i parameter_file.txt` in CASAL2. I digress, I highly recommend when estimating any model with CASAL2 that you try smaller values for the `tolerance` parameter. I have found for some models that if you make it say 0.00000002 that the solution is quite a bit different than when using the default (0.002). I believe this is not ideal behaviour from your model (Craig's Opinion) and that you should do some more investigative work on what parameters are causing the behaviour. An aside note: this will also effect your covariance matrix as with different tolerances and searchs you may end up with a different approximate hessian matrix which is inverted to solve for the covariance matrix. I thought that this might make a difference on MCMC results because the covariance is incorporated into the proposal distribution. However I have run a few MCMC's with varying tolerances and have found that doesn't effect the posterior distribution of the MCMC.

5.4.1. The numerical differences minimiser

The minimiser has three kinds of (non-error) exit status, depending on the minimiser:

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 the minimisers are local optimisation algorithms 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.

```
@minimiser numerical_diff
type numerical_differences
tolerance 1e-6
iterations 2500
evaluations 4000
```

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.

```
@minimiser DE_solver
type de_solver
tolerance 1e-6
iterations 2500
evaluations 4000
```

5.4.3. Betadiff minimiser

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

```
@minimiser beta_diff
type beta_diff
tolerance 1e-6
iterations 2500
evaluations 4000
```

5.4.4. ADOL-C minimiser

An auto-differentiable minimiser for non-linear models.

```
@minimiser ADOLC
type adolc
step_size 1e-6
iterations 2500
evaluations 4000
tolerance 1e-6
```

5.4.5. CPPAD minimiser

An auto-differentiable minimiser for non-linear models using the mumps solver, see `https://www.coin-or.org/CppAD/Doc/ipopt_solve.htm` for more information about this solver.

```
@minimiser CPPAD
type cppad
```

I have found this solver to be by far the quickest solver for models that have a reasonably well defined solution. What I mean by that, is there is 'good' information in the data to identify all the parameters. Now you may be thinking shouldn't this be the case for all minimisers? Short answer is yes, but the other minimisers are quicker than `cppad` to tell you there is not a reasonable solution.

5.4.6. Dlib minimiser

Non auto-diff minimiser

```
@minimiser Dlib
type dlib
tolerance 1e-6
iterations 2500
evaluations 4000
```

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

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 *n*th 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 *N*th element of the remainder is taken as the posterior sample. The chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

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

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

The user can specify the starting point of the point estimate minimiser using `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 is currently two options for the starting point of the Markov Chain:

- Start from the point estimate.
- Restart a chain given a covariance matrix and starting points (see section arg1)

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

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

by setting

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

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

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

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

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

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

1. You can request that the stepsize change adaptively at one or more sample numbers (See next paragraph for details on the stepsize adaptation methods)
2. You can request that the entire covariance matrix change adaptively at one or more sample numbers. At each adaptation, the covariance matrix is replaced with an empirical covariance, derived from the MCMC chain. The idea here is that an empirical covariance is a better approximation to the proposal distribution than the inverse of the hessian matrix, and can improve convergence and mixing of your chain.

The two methods that you can choose to adapt the step size are `double_half` or `ratio`, this is done through the input parameter `adapt_stepsize_method`. The `double_half` method is used in CASAL and (See Gelman et al. (Gelman et al., 1995) for justification). The algorithm for `double_half` is, 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. The `ratio` is taken from SPM. It adapts the current step size by, the acceptance rate since the last adaptation multiplied by 4.1667 to reach an acceptance rate of ≈ 0.24 see Sherlock and Roberts (2009) for justification on that acceptance rate.

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.

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

The 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.

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).

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.

Restarting a mcmc chain, in the case where computers get turned off and an mcmc execution is haltered. There is the ability to restart it from where it finishes.

```
casal2 -m --resume --objective-file Objective_file_name --sample-file Sample_file_name
```

where `Objective_file_name` is the file name containing the objective report and `Sample_file_name` is the file name containing the sample report from a mcmc chain.

The posterior sample can be used for (projections (Section 4.6)) or simulations (Section 6.6) with the values supplied using `casal2 -i` *`file`*.

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

Given a posterior (sub)sample, CASAL2 can calculate a list of output quantities for each sample point (see Section 7 scecifically tabular report). These quantities can be dumped into a file (using `casal2 -r --tabular`) 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 4.6). The advantage of this is that the parameter uncertainty, as expressed in your posterior distribution, can be included into the risk estimates.

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

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

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

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

3. Normal with mean μ and c.v. c

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

4. Normal with mean μ and standard deviation σ

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

5. Lognormal with mean μ and c.v. c

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

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

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

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

   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 $\tau\leq 0$.

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. Example of syntax to define the estimation of a parameter and the prior assumed follows;

```
## uniform-log example estimate
@estimate B0
type uniform_log # this command "type" defines the prior type.
parameter process[Recruitment].b0 # "Recruitment" is the label of your process
upper_bound 20000
lower_bound 1000

## Lognormal YCS estimation
@estimate year_class_strengths_1990_1995
type lognormal
parameter process[Recruitment].ycs_values{1990:1995}
#ycs_year   1990 1991 1992 1993 1994 1995
mu    1    1    1    1    1    1
cv  0.9  0.9  0.9  0.9  0.9  0.9
lower_bound 0.01 0.01 0.01 0.01 0.01 0.01
upper_bound 9 9 9 9 9 9
```

5.8. Penalties

Penalties are associated with processes and can be used to discourage parameter values or model outputs that are non-sensical, 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 remove or shift a *number* of individuals between categories or from the partition. For CASAL users many of the penalties that were available in CASAL have been moved to be additional priors, see Section 5.9.

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=mortality_instantaneous`, `@process[label].type=tag_by_length`, `@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. An example of applying a penalty

```
@process Mortality
type mortality_instantaneous
penalty CatchMustBeTaken

# define the penalty in an @penalty block
@penalty CatchMustBeTaken
type process
log_scale True
```

```
multiplier 10000
```

Penalties are added to the objective function in the following ways;

$$Penalty = (X_1 - X_2)^2 \tag{5.8}$$

or if `log_scale true`

$$Penalty = (log(X_1) - log(X_2))^2 \tag{5.9}$$

Where X_1 could be a known catch and X_2 is the model catch under a given set of parameter values. These are usually applied in situations where you have known numbers or weight. Another obvious example is tagging, we know for a fact that we tagged N fish this year so don't allow you model to apply less than that because that is inplausible.

5.9. Additional Priors

Additional priors can be thought of as the inverse of penalties. They constrain parameters in certain spaces. The types of additional priors available in CASAL2 are `vector_smoothing` and `vector_averaging`, defined as,

1. `vector_averaging`

   Applied to a vector parameter. Sum of squares of rth differences, optionally on a log scale. This encourages the vector to be like a polynomial of degree (r-1). Note a range of the vector to be smoothed can be specified (and if not, the smoother is applied to the entire vector), but this must be specified by an index of the vector and must be between 1 and the length of the vector, inclusive.

2. `vector_smoothing`

   Applied to a vector parameter. Square of (mean(vector)-k), or of (mean(log(vector))-l), or of (log(mean(vector)/m)). Encourages the vector to average arithmetically to k or m, or geometrically to exp(l). Typically used for YCS with k=1 or m=1 or l=0, to encourage the YCS to centre on 1. Optionally, you can choose to exclude indices outside a given set of bounds.

3. `lognormal` with mean μ and c.v. c

   $$-\log(\pi(p)) = \log(p) + 0.5\left(\frac{\log(p/\mu)}{s} + \frac{s}{2}\right)^2 \tag{5.10}$$

4. `uniform_log`

   $$-\log(\pi(p)) = \log(p) \tag{5.11}$$

5. `element_difference`

$$-\log(\pi(p)) = \log(p) \tag{5.12}$$

6. `Beta` Beta with mean μ and standard deviation σ, and range parameters A and B, for parameter value = p

$$-\log(\pi(p)) = (1-m)\log(p-A) + (1-n)\log(B-p) \tag{5.13}$$

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

5.9.1. Estimate Transformations

CASAL2 has multiple methods to transform a parameter some methods are for legacy purposes and some are more recent ideas. All transformations are implemented for the same reason and that is to try and achieve better optimisation from our models. Is is no surprise that complex population models can have highly correlated parameters so transforming them into orthogonal or into a different space is a way of trying to remove correlations and hope this allows our minimiser to find a 'global' minimum quicker. There are two main methods available, `transform_with_jacobian` and `transform_without_jacobian` but define the prior for the transformed variable. The latter has to more additional options where users can define the priors and bounds on either the transformed or non-transformed parameter. To read more about transformations and get a better understanding of why they are used we refer you to Gilks et al. (1995) specifically chapter 6.

5.9.1.1. Transform with Jacobian

The support of a random variable X with density $p_X(x)$ is that subset of values for which it has non-zero density,

$$supp(X) = \{x | p_X(x) > 0\} \tag{5.14}$$

If f is a transformation function defined on the support of X, then $Y = f(X)$ is a new random variable (transformed variable). This section shows the available transformations in CASAL2 and the probability density function of Y.

Suppose X is one dimensional and f: $supp(X) \rightarrow \mathbf{R}$ is a one-to-one, monotonic function with a differentiable inverse f^{-1}. Then the density of Y is given by

$$p_Y(y) = p_X(f^{-1}(y)) \left| \frac{\partial}{\partial y} f^{-1}(y) \right| \tag{5.15}$$

where $\left| \frac{\partial}{\partial y} f^{-1}(y) \right|$ is the jacobian term. The jacobian measures how the scale of the transformed variable changes with respect to the underlying variable. This can be expanded to the multivaiate case where the jacobian becomes a matrix of partial derivatives, see later for some example of multivariate cases. In equation 5.15 the term $p_X(f^{-1}(y)) = p_X(X)$ and in a bayesian context is the prior of the untransformed variable/parameter.

5.9.1.2. Transform without Jacobian

Specifying bounds on transformed variables

6. The observation section

6.1. Observations

The objective function is based on the goodness-of-fit of the model to your observations. Observations are typically supplied at an instance in time, over a group of aggregated categories. Most observations are different kinds of time series, i.e., data which were recorded for one or more years, in the same format each year. Examples of time series data types include relative abundance indices, commercial catch length frequencies, and survey numbers-at-age.

The definitions for each type of observation are described below, including how the observed values should be formatted, how CASAL2 calculates the expected values, and the likelihoods that are available for each type of observation.

There are two types of observations available in CASAL2. The first are observations that are associated with a **mortality block** and secondly observations that are associated with a specific process. These can be distinguished by the type definition. If an observation type begins with `process` it is an observation that is associated with a process. If a type does **not** begin with `process` it is associated with the mortality block of the time step that you define. For example the observation type `process_abundance` is a process based observation vs `process_abundance abundance`, which is an observation that is associated with a mortality block.

Process specific observations can also be broken into two types. **Specific process observations** are observations that are associated to a specific process (e.g. `process_proportions_migrating`), and **general process observations** are observations that can be associated with any process (e.g. `process_proportions_at_age`). These tiers of observations have been separated in different sections as to reduce the confusion.

6.1.1. Mortality block associated observations

All observations within this class are calculated in a similar fashion. That is an expectation is calculated at the beginning of the mortality block and at the end of the mortality block. CASAL2 then uses a linear interpolation to approximate an expectation part way through a mortality block usign the subcommand `time_step_proportion`. This could be useful if a survey occurs part-way through an exploitation phase. For example for if modelling a fish population this may be part-way through a fishing season. Each observation in this class will evaluate different expectations of the partition which will be explained in the following descriptions. A list of observation `types` that are available with this class of observations are as follows,

- `abundance`
- `biomass`
- `proportions_at_age`
- `proportions_at_length`
- `proportions_by_category`
- `tag_recapture_by_length`
- `tag_recapture_by_age`

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 aggregated categories. 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.

CASAL2 calculates the expected values by summing over the defined ages (via the age range and selectivity) and categories at both the beginning and end of a mortality block. You can prompt CASAL2 to approximate the expectation part way through the mortality block using the `time_step_proportion`. The default value CASAL2 uses us 0.5, which does linear interpolation between the start and end abundance (or biomass) from the mortality block.

For an abundance observation the expectation is calculated as follows,

$$E_{i,1} = \sum_{c=1} \sum_{a=1}^{A} S_{a,c} N_{a,c,i,1} \tag{6.1}$$

$$E_{i,2} = \sum_{c=1} \sum_{a=1}^{A} S_a N_{a,c,i,2} \tag{6.2}$$

Where $E_{i,1}$ is the expectation at the beginning of time step and $E_{i,2}$ is the expectation at the end of the time-step. S_a is the selectivity for age a and category c. If there is no mortality related to this observation then E_i which is used in the likelihood contribution is $E_{i,1}$. If this was a biomass observation we would replace $N_{a,c,i,1}$ in Equation (6.1) and (6.2) with $N_{a,c,i,1}\bar{w}_{a,c}$, where $\bar{w}_{a,c}$ is the mean weight of category c at age a. If the user wishes to apply 100% mortality then $E_i = E_{i,2}$. For applying quantities of mortality between these values (M_i), CASAL2 does the following linear interpolation.

$$E_i = |E_{i,1} - E_{i,2}| M_i \tag{6.3}$$

```
@observation MyAbundance
type abundance
years 1999
...
categories male
obs 1000
...
```

Or, for an observation aggregated over multiple categories,

```
@observation MyAbundance
type abundance
years 1990 1991
...
categories male+female
table obs
1990 1000
1991 1200
end_table
...
```

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

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

Proportions-at-age

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

The method of evaluating expectation are the same for all three of these sceneries. We will describe how you define these different scenarios and the expected dimensions of observation and error inputs that CASAL2 expects for each respective scenario with examples.

Like all types of observations that are associated with the mortality block, CASAL2 will evaluate the numbers at age before the mortality block (after taking into account a selectivity that the user defines) and after for the specified time step of the observation. CASAL2 will generate expectations from the partition part way through the mortality block using the subcommand `time_step_proportion`. This approximation is an linear interpolation of the numbers at age over the mortality block.

Once the interpolation is evaluated CASAL2 will apply ageing error if the user has specified it. CASAL2 finally converts numbers at age to proportions at age by dividing all numbers at age bin by the total and sending that to the likelihood to be evaluated.

Defining an observation for a single category 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.

```
@observation MyProportions
type proportions_at_age
...
categories male
min_age 3
max_age 9
years 1990
table obs
1990 0.01 0.09 0.20 0.20 0.35 0.10 0.05
end_table
...
```

Defining an observation for multiple categories extends on the single category implementation. 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 across all ages and categories.

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,

```
@observation MyProportions
type proportions_at_age
...
categories male female
min_age 1
max_age 5
years 1990 1991
table obs
1990 0.01 0.05 0.10 0.20 0.20 0.01 0.05 0.15 0.20 0.03
1991 0.02 0.06 0.10 0.21 0.18 0.02 0.03 0.17 0.20 0.01
end_table
...
```

Defining an observation across aggregated categories 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,

```
@observation MyProportions
type proportions_at_age
...
years 1990 1991
categories male+female
min_age 1
max_age 5
table obs
1990 0.02 0.13 0.25 0.30 0.30
1991 0.02 0.06 0.18 0.35 0.39
end_table
...
```

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
...
categories male_immature male_mature female_immature+female_mature
min_age 1
max_age 4
years 1990
table obs
year 1990 0.05 0.15 0.15 0.05 0.02 0.03 0.08 0.04 0.05 0.15 0.15 0.08
end_table
...
```

Proportions-at-length

Functionality regarding defining combinations of categories and aggregated categories directly translates over from proportions at age to proportions at length. The difference is the observation is over length bins instead of age-classes. CASAL2 calculates expectations of numbers at length by converting numbers at age to numbers by length by using the age-length relationship and distribution specified for the category specified in the `@age_length` block. Commands that are different are instead of supplying a minimum and maximum age users must supply a vector of length bins. If there is no plus group i.e `length_plus_group=false` CASAL2 expects a vector of proportions for each year that is $n-1$, where n is the number of lengths supplied. If `length_plus_group=true` CASAL2 expects a vector of proportions for each year that is n. The last proportion represents the numbers from the last length bin to the maximum length the age-length relationship allows.

```
@observation Observed_Length_frequency_Chat_east
type process_removals_by_length
years 1991 1992
likelihood multinomial
time_step Summer
fishery EastChathamRise
process instant_mort
categories male
length_plus_group false
length_bins 0 20 40 60 80 110
table obs
1991    0.2   0.25    0.15     0.2     0.2
1992    0.12  0.25    0.28    0.25    0.1
end_table
table error_values
1991 25
1992 37
end_table
```

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).

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 i.e.,

```
@observation MyProportions
type proportions_by_category
years 1990 1991
...
categories male
categories2 female
min_age 1
max_age 5
table obs
1990 0.01 0.05 0.10 0.20 0.20
1991 0.02 0.06 0.10 0.21 0.18
end_table
...
```

Tag Recapture by length

Tag data is primarily used to estimate the population abundance of fish. In some models, this estimation can only be made outside the model and the result is used as an estimate of abundance in the model. But in CASAL the tagging data can, alternatively, be fitted within the model.

Before adding a tag-recapture time series, you will need to define a tag-release process (Section 4.7.5). Tagging events list the labels of the tags which are modelled, and define the events where fish are tagged (i.e., CASAL2moves fish into the section of the partition corresponding to a specific tag).

The observations are divided into two parts: (i) the number of fish that were scanned, and (ii) the number of tags that were recaptured. Each can be specified by categories, or for combinations of categories. The precise content of the scanned and recaptured observations depends on the sampling method, and the available options are:

1. age: both scanned and recaptured are vectors containing numbers-at-age. Only available in an age-based model. The selectivity ogive is redundant and cannot be supplied.
2. size: both scanned and recaptured are vectors containing numbers-at-size. Can be used in either an age- or size-based model. The selectivity ogive is redundant and cannot be supplied.

When defining the tag-recapture time series, you also need to specify:

- the time step,
- the years (unlike a tag-release process, the tag-recapture observations can occur over several years),
- the probability that each scanned tagged fish is detected as tagged (may be less than 1 if the observers are not infallible). The expected number of tags detected is calculated by multiplying this number by the number of tagged fish in the sample,
- the tagged category or categories (Make up the recaptures),
- the categories scanned (All the fish sampled for tags),
- A selectivity used in the recapture process,
- the size classes if the observations are size-based in an age-based model.

An example of a tag recapture observation applied in CASAL2 is shown, below

```
## For the following partition
@categories
format sex.area.tag
names   male.Area1.2011,notag female.Area1.2011,notag

@observation Tag_2011_Area1_recap_2012 ## individuals tagged in 2011 and recaptured in 2012
## in Area1
type tag_recapture_by_length
categories *.Area1.2011+  ## male and femaled tagged categories
categories2 format=*.Area1.*+ ## scanned categories in Area1
detection 0.85 ## detection probability
likelihood binomial ## likelihood choice
selectivities One ## label of selectivity for tagged
selectivities2 One ## label of selectivity for scanned
years 2012  ## years to apply observation
time_step step2  ## time_step to apply observation
time_step_proportion 0.5 ## proportion of mortality applied before observation is calculated
length_bins 21 30 40 50 ## size bins
plus_group true ## is the last bin a plus group i.e. 50cm +

table scanned
2012 281271 41360 30239 12234
end_table

table recaptured
2012 15 20 12 2
end_table

delta 1e-11 ## robustification value
dispersion 6.3 ## dispersion factor
```

The tag-recapture likelihood (binomial) is specified below as it is a modified version of the more general binomial. Note that this likelihood does not have any user-set precision parameters such as *N* or *c.v.* (though there are user-specified robustification and dispersion parameters available). Note that factorials are calculated using the log-gamma function, to allow for non-integer arguments where necessary (and avoid overflow errors).

6.1.2. General process observations

A list of `types` that are associated with this set of observations.

- `process_abundance`
- `process_biomass`
- `process_proportions_at_age`
- `process_proportions_at_length`
- `process_proportions_by_category`

These observations have the same expectations as the mortality block versions described in Section 4.4. With the exception that instead of wrapping a mortality block they can wrap any process type available in CASAL2.

6.1.3. Specific process observations

A list of `types` that are associated with this set of observations are;

- `process_removals_by_age`,
- `process_removals_by_length`,
- `process_proportions_migrating`.

Process removals by age

Removals at age observations are observations of the relative number of individuals at age, partway through a process of type `mortality_instantaneous`. This observation is exclusively associated with the process of type `mortality_instantaneous`, and will error out if associated with any other process type.

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 that is associated with the process).

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.

The expectations from this observation are generated whilst the process is being executed. The expectation of numbers at age a for category c from exploitation method m ($E[N_{a,c,m}]$) are defined as,

$$E[N_{a,c,m}] = N_{a,c}U_{a,m}S_{a,c,m}0.5M_{a,c} \tag{6.4}$$

where, $N_{a,c}$ are the numbers at age in category c before the process is executed, $U_{a,m}$ is the exploitation rate for age a from method m. $S_{a,c,m}$ is the selectivity and M is the natural mortality. These are all relevant to the time step which the user defines.

The observation class then acquires the variable $E[N_{a,c,m}]$ and applies ageing error if the user has specified it. Then it amalgamates the observations by method and category depending on how the user specifies the observation, before converting numbers at age to proportions and sending them to the likelihood to be evaluated.

Likelihoods that are available for this observation class are the mulitnomial, dirichlet and the lognormal. See Section 6.2 for information on the respected likelihood.

Process removals by length

Removals by length observations are observations of the relative number of individuals at length, partway through a process of type `mortality_instantaneous`. This observation is exclusively associated with the process of type `mortality_instantaneous`, and will error out if associated with any other process type.

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 that is associated with the process).

The expectations from this observation are generated whilst the process is being executed. The expectation of numbers at age a for category c from exploitation method m ($E[N_{a,c,m}]$) are defined as,

$$E[N_{a,c,m}] = N_{a,c}U_{a,m}S_{a,c,m}0.5M_{a,c} \tag{6.5}$$

where, $N_{a,c}$ are the numbers at age in category c before the process is executed, $U_{a,m}$ is the exploitation rate for age a from method m. $S_{a,c,m}$ is the selectivity and M is the natural mortality. These are all relevant to the time step which the user defines.

The observation class acquires the variable $E[N_{a,c,m}]$ from the process and applies the age-length relationship specified in the model. This converts numbers at age to numbers at age and length, where CASAL2 then converts to numbers at length. Then it amalgamates the observations by method and category depending on how the user specifies the observation, before converting numbers at age to proportions and sending them to the likelihood to be evaluated.

```
@observation observation_fishery_LF
type process_removals_by_length
...
years  1993 1994 1995
method_of_removal FishingEast
mortality_instantaneous_process instant_mort
length_plus_group false
length_bins 0 20 40 60 80 110
delta 1e-5
table obs
1993    0.0   0.05    0.05    0.10    0.80
1994    0.05  0.1     0.05    0.05    0.75
1995    0.3   0.4     0.2     0.05    0.05
end_table

table error_values
1993 31
1994 34
1995 22
end_table
```

Likelihoods that are available for this observation are the mulitnomial, dirichlet and the lognormal. See Section 6.2 for information on the respected likelihood.

Proportions migrating

This observation is of the proportion migrating from one area to another. This observation is exclusively associated with the process type `transition_category`, and will error out when trying to associate with any other process type. This observation is used to inform migration rates in migration processes. This observation class is used in the Hoki stock assessment see Francis et al. (2003) for more information on how these observations are collected and the situation you would use it. This observation calculates an expectation E_a of proportions for each age class a that have migrated, by evaluating the following,

$$E_a = \frac{N_a - N'_a}{N_a} \tag{6.6}$$

where, N_a are the numbers of individuals in age a before the migration process occurs and N'_a is the number of individuals after the migration process occurs.
The likelihoods that are allowed for this observation are the lognormal, multinomial and dirichlet.

An extract of the Hoki stock assessment is as follows,

```
@observation pspawn_1993
type process_proportions_migrating
years 1993
time_step step4
process Wspmg ## migration process that the observation is associated with
age_plus true
min_age 4
max_age 9
likelihood lognormal
categories male.west+female.west ## Categories to evaluate the prportion for
ageing_error Normal_offset ## label for an @ageing_error block
table obs
#age    4    5    6    7    8    9
1993 0.64 0.58 0.65 0.66 0.71 0.60
end_table

table error_values
## if lognormal these are c.v.'s
1993 0.25
end_table
```

6.2. Likelihoods

6.2.1. Likelihoods for proportions-at-age observations

CASAL2 implements three likelihoods for proportions-at-age observations, the multinomial likelihood, dirichlet, 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;

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(Z(E_i, \delta)) \tag{6.7}$$

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{6.8}$$

The default value of δ is 1×10^{-11}.

The dirichlet likelihood

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

$$-\log(L) = -\log(\Gamma\sum_i(\alpha_i)) + \sum_i \log(\Gamma(\alpha_i)) - \sum_i(\alpha_i - 1)\log(Z(O_i, \delta)) \tag{6.9}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.10}$$

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;

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i/Z(E_i,\delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{6.11}$$

where

$$\sigma_i = \sqrt{\log\left(1 + c_i^2\right)} \tag{6.12}$$

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,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.13}$$

The default value of δ is 1×10^{-11}.

6.2.2. Likelihoods for abundance and biomass observations

Abundance and biomass observations are expected as an annual time series in CASAL2, where they select the same categories over that time series. The parameters and inputs needed to use this observation class are: a observation O_i, c.v. c_i, catchability coefficient q, where i indexed the year. CASAL2 calculates an expectation E_i and scales it by q before comparing it to O_i. This means that the value chosen for q will determine whether the observation is relative ($q \neq 1$) or absolute $q = 1$. Before we describe each of the likelihoods we will discuss the methods available to handle $q's$:

1. The $q's$ can be treated as nuisance parameters. For each set of values of the free parameters, the model uses the values of the $q's$which minimise the objective function. These optimal $q's$ are calculated algebraically (see Section 6.4). If one of the $q's$ falls outside the bounds specified by the user, it is set equal to the closest bound. This approach reduces the size of the parameter vector and hence should improve the performance of the estimation method. However, it is not correct when calculating a sample from the posterior in a Bayesian analysis (except asymptotically, see Walters and Ludwig (1994)) and we offer the following alternative;
2. The $q's$ can be treated as ordinary free parameters.

For both options, it is necessary to evaluate the contribution of O_i to the negative loglikelihood for a given value of q. Each observation O_i varies about qE_i express the variability of O_i in terms of its c.v. c_i (or in one case, its standard deviation si). Here are the likelihoods, which are expressed on the objective-function scale of -log(L):

The lognormal likelihood

The negative log likelihood for a the lognormal is as follows,

$$-\log(L)=\sum_i\left(\log(\sigma_i)+0.5\left(\frac{\log(O_i/qZ(E_i,\delta))}{\sigma_i}+0.5\sigma_i\right)^2\right) \tag{6.14}$$

where

$$\sigma_i=\sqrt{\log\left(1+c_i^2\right)} \tag{6.15}$$

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

This reflects the distributional assumptions that O_i has the lognormal distribution, that the mean of O_i is qE_i and the c.v. of O_i is c_i.

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

The default value of δ is 1×10^{-11}.

The normal likelihood

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

$$-\log(L)=\sum_i\left(\log(c_iE_i)+0.5\left(\frac{O_i-E_i}{Z(c_iE_i,\delta)}\right)^2\right) \tag{6.17}$$

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

$$Z(\theta,\delta)=\begin{cases}\theta, & \text{where } \theta\geq r\\ \delta/(2-\theta/\delta), & \text{otherwise}\end{cases} \tag{6.18}$$

The default value of δ is 1×10^{-11}.

This reflects the distributional assumptions that O_i has the normal distribution, that the mean of O_i is qE_i and the c.v. of O_i is c_i.

6.2.3. Likelihoods for tag recapture by age and length observations

The binomial likelihood

Designed for situations where the size frequencies or age frequencies of the recaptured tagged fish and of the scanned fish are known. Available in both age or size based models.

Here we define the likelihood as a binomial, but based on sizes, rather than ages,

$$\begin{aligned}-\log(L)' = -\sum_i [\, &\log(n_i!) - \log((n_i - m_i)!) - \log((m_i)!) + m_i \log\left(Z\left(\frac{M_i}{N_i}, \delta\right)\right) \\ &+ (n_i - m_i) \log\left(Z\left(1 - \frac{M_i}{N_i}, \delta\right)\right) \,]\end{aligned} \tag{6.19}$$

where
n_i = number of fish at size or age i that were scanned
m_i = number of fish at size or age i that were recaptured
N_i = number of fish at size or age i in the available population (tagged and untagged)
M_i = number of fish at size or age i in the available population that have the tag after a detection probability p_d has been applied, $M_i = M_i' p_d$, where M_i' is the expected available population that have the tag.

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

$$Z(x,\delta) = \begin{cases} x & \text{where } x \geq \delta \\ \frac{\delta}{(2 - x/\delta)} & \text{otherwise} \end{cases}$$

Finally if a dispersion parameter (τ) is described in the observation then the final negative log likelihood $-log(L)$ contribution is,

$$-log(L) = -log(L)'/\tau$$

6.2.4. 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;

$$\begin{aligned}-\log(L) = -\sum_i [\, &\log(N_i!) - \log((N_i(1 - O_i))!) - \log((N_i O_i)!) + N_i O_i \log(Z(E_i, \delta)) \\ &+ N_i(1 - O_i) \log(Z(1 - E_i, \delta)) \,]\end{aligned} \tag{6.20}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.21}$$

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;

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{6.22}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{6.23}$$

The default value of δ is 1×10^{-11}.

6.3. 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

$$c'_i = \sqrt{c_i^2 + c_{process\_error}^2} \tag{6.24}$$

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,

$$N'_i = \frac{1}{1/N_i + 1/N_{process\_error}} \tag{6.25}$$

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.

6.4. Calculating nuisance q's

This section describes the theory used to calculate nuisance (analytical) catchability coefficients *qs* (see Section 6.2.2). From the user's point of view, the essence is that you can use nuisance *qs* in the following situations:

1. With maximum likelihood.
2. With Bayesian estimation, providing that your provide an additional prior on the q is one of the following:
    - none (default)
    - Uniform-log
    - Lognormal with observations distributed lognormal, robustified lognormal

Table 6.1 displays the scenarios when the nuisance catchability can be used for a Bayesian analysis.

Table 6.1: Equations used to calculate nuisance q's. (*=no analytic solution found.)

Distribution of observations	Maximum Likelihood	None	Uniform-log	Normal	lognormal
Normal	(6.26)	(6.26)	(6.28)	*	*
Lognormal	(6.29)	(6.29)	(6.33)	*	(6.34)

Note that *qs* are calculated for robustified lognormal likelihoods as if they were ordinary lognormal likelihoods.

The equations and their derivations follow. Let $\sigma_i = \sqrt{log(1+c_i^2)}$ throughout, and let n be the number of observations in the time series. The case of multiple time series sharing the same q, and the modifications required for the assumption of curvature, are addressed at the end of this subsection.

First, consider maximum likelihood estimation. When the (Oi) are assumed to be normally distributed,

$$-log(L) = \sum_i log(c_i q E_i) + 0.5 \sum_i \left(\frac{O_i - qE_i}{c_i q E_i} \right)^2 \tag{6.26}$$

The value of q which minimises the objective function is found by solving for q under the following condition, $\partial/\partial q(-log(L)) = 0$

$$\frac{\partial}{\partial q}(-log(L)) = \frac{n}{q} + \frac{1}{q^2} \sum_i \frac{O_i}{c_i^2 E_i} - \frac{1}{q^3} \sum_i \left(\frac{O_i}{c_i E_i} \right)^2 \tag{6.27}$$

hence

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4nS_2}}{2n} \tag{6.28}$$

where $S_1 = \sum_i (O_i/c_i^2 E_i)$ and $S_2 = \sum_i (O_i/c_i E_i)^2$

When the (O_i) are assumed to be lognormally distributed,

$$-log(L) = \sum_i log(\sigma_i) + 0.5 \sum_i \left(\frac{log(O_i) - log(qE_i) + 0.5\sigma_i^2}{\sigma_i} \right)^2 \tag{6.29}$$

$$\frac{\partial}{\partial q}(-log(L)) = \frac{-1}{q} \sum_i \left(\frac{log(O_i/E_i) - log(q) + 0.5\sigma_i^2}{\sigma_i^2} \right) \tag{6.30}$$

$$\hat{q} = exp\frac{0.5n + S_3}{S_4} \tag{6.31}$$

where $S_3 = \sum_i(log(O_i/E_I)/\sigma_i^2)$ and $S_4 = \sum_i(1/\sigma_i^2)$

Next consider Bayesian estimation, where we must also specify a prior for q.

The effects of the prior on the equations are to replace likelihood L by posterior P throughout, to add $-log(\pi(q))$ to the equation for $-log(P)$ and $\partial/\partial q(-log(-\pi(q)))$ to the equation for $\partial/\partial q(-log(P))$

This last term is 0 for a uniform prior on q, $1/q$ for a log-uniform prior, and $\frac{1}{q}\left(1.5 + \frac{log(q)-log(\mu_q)}{\sigma_q^2}\right.$ for a lognormal prior,

where μ_q and c_q are the mean and c.v of the prior on q and $\sigma_q = \sqrt{log(1+c_q^2)}$. Clearly, if the prior is uniform, the equation for $\hat{q}$ is teh same as teh maximum likelihood estimation.

When the (O_i) are assumed to be normally distributed and teh prior is log-uniform equation (6.28) becomes,

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4(n+1)S_2}}{2(n+1)} \tag{6.32}$$

but we cannot solve for $\hat{q}$ with either a normal or lognormal prior.

When the O_i are assumed to be lognormally distributed and the prior is log-uniform, equation (6.31) becomes

$$\hat{q} = exp\frac{0.5n - 1 + S_3}{S_4} \tag{6.33}$$

and if the prior is lognormal,

$$\hat{q} = exp\frac{0.5n - 1.5 + log(\mu_q)/\sigma_q^2 + S_3}{S_4 + 1/\sigma_q^2} \tag{6.34}$$

but it is not possible to solve for $\hat{q}$ with a normal prior. An example of specifying the syntax and an equivalent additional prior see below

```
@catchability chatTANq
type nuisance
upper_bound 0.6
lower_bound 0.0001

@additional_prior chatTANq_prior
type lognormal
parameter catchabilityp[chatTANq].q
mu 0.3
cv 0.2
```

6.5. Ageing error

CASAL2 can apply ageing error to expected age frequency generated by the model. The ageing error is applied as a misclassification matrix, which has the effect of 'smearing' the expected age frequencies. This is mimicking the error involved in identifying the age of individuals. For example fish species are aged by reading the ear bones (otoliths) which can be quite difficult depending on the species. 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 7.12 for reporting the misclassification matrix at the end of model run.

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.

6.6. Simulating observations

CASAL2 can generate simulated observations for a given model with given parameter values using `casal2 -s 1` (To simulate one set of simulated observations). 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

An important note when simulating: CASAL2 will **not** automatically report simulated observations when users undertake a `casal2 -s 1` run, you must write an explicit report using the `simulated_observation` report (`@report[label].type=observation`). See Section 7 for more information on how to write this report.

6.7. 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.

6.8. Residuals

CASAL2 will only print the usual residual (i.e. observed less fitted) using the report type `@report`.type=observation. For an observation O and F the corresponding fit (=qE for relative observations), then

- Residuals = O - F

Pearson and Normalised residuals can be generated using CASAL2 **R** package with-in the **R** environment. For specific R functions see Section 14. The definitions used in the calculations are as follows,

1. *Pearson residuals* attempt to express the residual relative to the variability of the observation, and are defined as (O-F)/std.dev.(O), where std.dev.(O) is calculated as
   - F $\times$ cv for normal, lognormal, robustified lognormal, and normal-log error distributions.
   - s for normal-by-standard deviation error distributions.
   - $\sqrt{\frac{Z(F,r)(1-Z(F,r))}{N}}$ for multinomial or binomial likelihoods.
   - $\sqrt{\frac{(F+r)(1-F+r)}{N}}$ for binomial-approx likelihood likelihoods.
2. *Normalised residuals* to express the residual on a standard normal scale, and are defined as:
   - Equal to the Pearson residuals for normal error distributions.
   - $(\log(O/F)+0.5\sigma^2)/\sigma$ for lognormal (including robustified lognormal) error distributions, where $\sigma = \sqrt{log(1+cv^2)}$.

- log(O/F)/σ for normal-log error distributions, again with $\sigma = \sqrt{log(1+cv^2)}$.
- And are otherwise undefined.

where $Z(F,r)$ is the robustifying term on F (fit or expectation of the observation). This robustifying is described earlier in the likelihood section.

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 of report types, 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 aster-ix 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 within an **R** environment.

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 filename` command. For estimate values in standard output users are recommended to use `type=estimate_value`.

Reports can be defined in an `@report` but 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. Certain reports are associated with certain CASAL2 run modes. 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 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. This report will print out in the following runmodes `-r, -e, -f`.

7.2. 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). This report will print out in the following runmodes `-r, -e, -f`.

7.3. Print the age length and length weight values

Print the length and weight for an age of the partition for a given year or given years and time-step. This prints out, the length and weight value for each age class and category in the partition for each year and time step. Note that this report is evaluated at the end of the time-step in the given year(s). This report will print out in the following runmodes `-r, -e, -f`.

```
@report length_weight_at_age
type partition_mean_weight
time_step step2
years 1900:2013
```

7.4. 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. This report will print out in the following runmodes `-r, -e, -f`.

7.5. 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. This report will print out in the following runmodes `-r, -e, -f`.

7.6. Print the estimated parameters

Print a summary of the estimated parameters using the following type `estimate_summary`, including the parameter name, lower and upper bounds, the label of the prior, and its value. This report will print out in the following runmodes `-r, -e`.

7.7. 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. This report will print out in the following runmodes `-r, -e`.

7.8. 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. This report will print out in the following runmodes `-r, -e, -f`.

7.9. Print the covariance matrix

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

7.10. 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.

```
@report Tan_at_age_obs
type observation
observation TAN_AT_AGE
```

7.11. 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. This report will print out in the following runmodes `-s`.

7.12. Print the ageing error misclassification matrix

Prints out the ageing error misclassification matrix used to offset observations within during model the model fitting procedure.

7.13. 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.

7.14. 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.

7.15. Print the results of an MCMC

Print the MCMC samples, objective function values, and proposal covariance matrix following an MCMC. This report will print out in the following runmode `-m`.

7.16. 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. This report will print out by default using the following runmodes `-m`.

7.17. 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. This report will print out by default using the following runmodes `-m`.

7.18. Print time varying parameters

Print all `@time_varying` blocks with the values and years that they were implemented in. This report will print out in the following runmodes `-r, -e, -m`.

```
@report time_varying_parameters
type time_varying
```

7.19. 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 sample or -o outputs). Tabular reports will print out a row that will correspond with each row of the `-i` input files. Tabular reporting is invoked at the command line using the `--tabular` command e.g. (`casal2 -r --tabular -i file_name`). Currently derived quantities, processes, observations 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. This command is the same as running `casal -v` in CASAL.

8. Population command and subcommand syntax

8.1. Model structure

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

`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

`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

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

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

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

`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`

`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`

`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.

`length_bins`
Type: constant vector
Default: true

`base_weight_units` Define the units for the base weight. This will be the default unit of any weight input parameters
Type: string
Default: tonnes
Allowed Values: grams, tonnes, kgs

8.2. Initialisation

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

`label` The label of the initialisation phase
Type: string
Default: No Default

`type` The type of initialisation
Type: string
Default: iterative

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

`categories` The list of categories for the Cinitial initialisation
Type: string vector
Default: No Default

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

`insert_processes` Additional processes not defined in the annual cycle, that are to beinserted into this initialisation phase
Type: string vector
Default: true

`exclude_processes` Processes in the annual cycle to be excluded from this initialisation phase
Type: string vector
Default: true

`casal_intialisation_switch` Run an extra annual cycle to evalaute equilibrium SSB's. Warning - if true, this may not correctly evaluate the equilibrium state. Use true if attempting to replicate a legacy CASAL model
Type: boolean
Default: false

8.2.3. `@initialisation_phase[label].type=iterative`

`years` The number of iterations (years) over which to execute this initialisation phase
Type: non-negative integer
Default: No Default

`insert_processes` (years) over which to execute this initialisation phase
Type: string vector
Default: true

`exclude_processes` Processes in the annual cycle to be excluded from this initialisation phase
Type: string vector
Default: true

`convergence_years` The iteration (year) when the test for converegence (lambda) is evaluated
Type: non-negative integer vector
Default: true

`lambda` The maximum value of the absolute sum of differences (lambda) between the partition at year-1 and year that indicates successfull convergence
Type: constant
Default: 0.0

8.2.4. `@initialisation_phase[label].type=state_category_by_age`

`categories` The list of categories for the category state initialisation
Type: string vector
Default: No Default

`min_age` The minimum age of values supplied in the definition of the category state
Type: non-negative integer
Default: No Default

`max_age` The minimum age of values supplied in the definition of the category state
Type: non-negative integer
Default: No Default

8.3. Categories

`@categories` *`label`* Define an object of type *categories*

`format` The format that the category names 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

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

8.4. Time-steps

`@time_step` *`label`* Define an object of type *time_step*

`label` The label of the timestep
Type: string
Default: No Default

`processes` The labels of the processes for this time step in the order that they occur
Type: string vector
Default: No Default

8.5. Processes

`@process` *`label`* Define an object of type *process*

`label` The label of the process
Type: string
Default: No Default

`type` The type of process
Type: string
Default: ""

`print_report` Indicates if a process report should be generated for this process
Type: boolean
Default: false

8.5.1. `@process[label].type=ageing`

`categories` The labels of the categories
Type: string vector
Default: No Default

8.5.2. `@process[label].type=maturation`

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

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

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

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

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

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

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

`m` Mortality rates
Type: estimable
Default: No Default

`time_step_ratio` Time step ratios for the mortality rates
Type: constant vector
Default: true

`selectivities` List of selectivities for the categories

Type: string vector
Default: No Default

8.5.4. **`@process[label].type=mortality_event`**

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

`years` Years in which to apply the mortality process
Type: non-negative integer vector
Default: No Default

`catches` The number of removals (catches) to apply for each year
Type: estimable vector
Default: No Default

`u_max` Maximum exploitation rate ($Umax$
Type: estimable
Default: 0.99

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

`penalty` The label of the penalty to apply if the total number of removals cannot be taken
Type: string
Default: ””

8.5.5. **`@process[label].type=mortality_event_biomass`**

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

`selectivities` The labels of the selectivities for each of the categories
Type: string vector
Default: No Default

`years` Years in which to apply the mortality process
Type: non-negative integer vector
Default: No Default

`catches` The biomass of removals (catches) to apply for each year

Type: estimable vector
Default: No Default

`u_max` Maximum exploitation rate (*Umax*
Type: estimable
Default: 0.99

`penalty` The label of the penalty to apply if the total biomass of removals cannot be taken
Type: string
Default: ””

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

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

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

`is_abundance` Is vulnerable amount of prey and predator an abundance [true] or biomass [false]
Type: boolean
Default: true

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

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

`x` This parameter controls the type of functional form, Holling function type 2 (x=2) or 3 (x=3), or generalised (Michaelis Menten, x,=1
Type: estimable
Default: No Default
Lower Bound: 1.0 (inclusive)

`u_max` Maximum exploitation rate (*Umax*

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

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

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

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

`years` Years in which to apply the mortality process
Type: non-negative integer vector
Default: No Default

8.5.7. `@process[label].type=mortality_initialisation_event`

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

`catch` The number of removals (catches) to apply for each year
Type: estimable
Default: No Default

`u_max` Maximum exploitation rate ($Umax$
Type: estimable
Default: 0.99

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

`penalty` The label of the penalty to apply if the total number of removals cannot be taken
Type: string
Default: ””

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

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

`catch` The number of removals (catches) to apply for each year
Type: estimable
Default: No Default

`u_max` Maximum exploitation rate (*Umax*
Type: estimable
Default: 0.99

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

`penalty` The label of the penalty to apply if the total number of removals cannot be taken
Type: string
Default: ””

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

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

`m` Natural mortality rates for each category
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

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

`selectivities` The selectivities to apply on the categories for natural mortality
Type: string vector
Default: No Default

8.5.10. `@process[label].type=mortality_prey_suitability`

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

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

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

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

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

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

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

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

`years` Year that process occurs
Type: non-negative integer vector
Default: No Default

8.5.11. `@process[label].type=nop`

8.5.12. `@process[label].type=recruitment_beverton_holt`

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

`r0` R0
Type: estimable
Default: false

`b0` B0
Type: estimable
Default: false

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

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

`ssb_offset` Spawning biomass year offset
Type: non-negative integer
Default: true

`steepness` Steepness
Type: estimable
Default: 1.0

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

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

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

`ycs_years` Recruitment years. A vector of years that relates to the year of the spawning event

that created this cohort
Type: non-negative integer vector
Default: false

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

8.5.13. `@process[label].type=recruitment_constant`

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

`proportions` Proportions
Type: estimable
Default: true

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

`r0` R0
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

8.5.14. `@process[label].type=survival_constant_rate`

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

`s` Survival rates
Type: estimable
Default: No Default

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

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

8.5.15. `@process[label].type=tag_by_age`

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

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

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

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

`penalty` Penalty label
Type: string
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

`initial_mortality`
Type: constant
Default: 0

`initial_mortality_selectivity`
Type: string
Default: ""

`loss_rate`
Type: constant vector
Default: No Default

`loss_rate_selectivities`

Type: string vector
Default: true

`selectivities`
Type: string vector
Default: No Default

`n`
Type: constant vector
Default: true

8.5.16. `@process[label].type=tag_by_length`

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

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

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

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

`penalty` Penalty label
Type: string
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

`initial_mortality`
Type: constant
Default: 0

`initial_mortality_selectivity`
Type: string
Default: ””

`selectivities`
Type: string vector
Default: No Default

`n`
Type: constant vector
Default: true

8.5.17. **`@process[label].type=tag_loss`**

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

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

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

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

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

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

8.5.18. **`@process[label].type=transition_category`**

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

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

`proportions` Proportions
Type: estimable
Default: No Default

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

8.5.19. `@process[label].type=transition_category_by_age`

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

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

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

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

`penalty` Penalty label
Type: string
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 of type *time_varying*

`label` The time-varying label
Type: string
Default: No Default

`type` The time-varying type
Type: string
Default: ””

`years` Years in which to vary the values
Type: non-negative integer vector
Default: No Default

`parameter` The name of the parameter to time vary
Type: string
Default: No Default

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

`values`
Type: constant vector
Default: No Default

`a`
Type: constant
Default: No Default

`b`
Type: constant
Default: No Default

`c`
Type: constant
Default: No Default

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

8.6.2. `@time__varying[label].type=constant`

`values` Value to assign to addressable
Type: estimable vector
Default: No Default

8.6.3. `@time__varying[label].type=exogenous`

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

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

8.6.4. `@time__varying[label].type=linear`

`slope` The slope of the linear trend (additive unit per year
Type: estimable
Default: No Default

`intercept` The intercept of the linear trend value for the first year
Type: estimable
Default: No Default

8.6.5. `@time__varying[label].type=random_draw`

`mean` Mean
Type: estimable
Default: 0

`sigma` Standard deviation
Type: estimable
Default: 1

`distribution` distribution
Type: string
Default: normal
Allowed Values: normal, lognormal

8.6.6. `@time__varying[label].type=random_walk`

`mean` Mean
Type: estimable
Default: 0

`sigma` Standard deviation
Type: estimable
Default: 1

`upper_bound` Upper bound for the random walk
Type: constant
Default: 1

`upper_bound` Lower bound for the random walk
Type: constant
Default: 1

`rho` Auto Correlation parameter
Type: constant
Default: 1

`distribution` distribution
Type: string
Default: normal

8.7. Derived quantities

`@derived_quantity` *`label`* Define an object of type *derived_quantity*

`label` Label of the derived quantity
Type: string
Default: No Default

`type` Type of derived quantity
Type: string
Default: No Default

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

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

`selectivities` A list of one selectivity
Type: string vector
Default: No Default

`time_step_proportion` Proportion through the mortality block of the time step when calculated
Type: constant
Default: 0.5
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`time_step_proportion_method` Method for interpolating for the proportion through the mortality block
Type: string
Default: weighted_sum
Allowed Values: weighted_sum, weighted_product

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

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

8.8. Age-length relationship

`@age_length` *`label`* Define an object of type *age_length*

`label` Label of the age length relationship
Type: string
Default: No Default

`type` Type of age length relationship
Type: string
Default: No Default

`time_step_proportions` the fraction of the year applied in each time step that is added to the age for the purposes of evaluating the length, i.e., a value of 0.5 for a time step will evaluate the length of individuals at age+0.5 in that time step
Type: constant vector
Default: true

`distribution` The assumed distribution for the growth curve
Type: string
Default: normal

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

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

`casal_switch` If true, use the (less accurate) equation for the cumulative normal function as was used in the legacy version of CASAL.
Type: boolean
Default: false

8.8.1. `@age_length[label].type=data`

`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` The label from an associated length-weight block
Type: string
Default: No Default

`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

8.8.2. `@age_length[label].type=none`

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

`y1` Define the y1 parameter of the Schnute relationship
Type: estimable
Default: No Default

`y2` Define the y2 parameter of the Schnute relationship
Type: estimable
Default: No Default

`tau1` Define the τ_1 parameter of the Schnute relationship
Type: estimable
Default: No Default

`tau2` Define the τ_2 parameter of the Schnute relationship
Type: estimable
Default: No Default

`a` Define the a parameter of the Schnute relationship
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)

`b` Define the b parameter of the Schnute relationship
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

`length_weight` Define the label of the associated length-weight relationship
Type: string
Default: No Default

`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

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

`linf` Define the $L_{infinity}$ parameter of the von Bertalanffy relationship
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)

`k` Define the k parameter of the von Bertalanffy relationship
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)

`t0` Define the t_0 parameter of the von Bertalanffy relationship
Type: estimable
Default: No Default

`length_weight` Define the label of the associated length-weight relationship

Type: string
Default: No Default

`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

8.9. Length-weight

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

`label` The label of the length-weight relationship
Type: string
Default: No Default

`type` The type of the length-weight relationship
Type: string
Default: No Default

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

`a` The *a* parameter in the basic length-weight relationship
Type: constant
Default: No Default

`b` The *b* parameter in the basic length-weight relationship
Type: constant
Default: No Default

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

8.9.2. `@length__weight[label].type=none`

8.10. Selectivities

`@selectivity` *`label`* Define an object of type *selectivity*

`label` The label for this selectivity
Type: string
Default: No Default

`type` The type of selectivity
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

8.10.1. `@selectivity[label].type=all_values`

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

8.10.2. `@selectivity[label].type=all_values_bounded`

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

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

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

8.10.3. `@selectivity[label].type=constant`

`c` C
Type: estimable
Default: No Default

8.10.4. `@selectivity[label].type=double_exponential`

`x0` X0
Type: estimable
Default: No Default

`x1` X1
Type: constant
Default: No Default

`x2` X2
Type: constant
Default: No Default

`y0` Y0
Type: estimable
Default: No Default

`y1` Y1
Type: estimable
Default: No Default

`y2` Y2
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

8.10.5. `@selectivity[label].type=double_normal`

`mu` Mu
Type: estimable
Default: No Default

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

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

`alpha` Alpha

Type: estimable
Default: 1.0

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

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

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

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

`alpha` Alpha
Type: constant
Default: 1.0

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

`a50` A50
Type: estimable
Default: No Default

`ato95` aTo95
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

8.10.8. `@selectivity[label].type=knife_edge`

`e` Edge
Type: estimable
Default: No Default

`alpha` Alpha

Type: estimable
Default: 1.0

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

`a50` A50
Type: estimable
Default: No Default

`ato95` Ato95
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

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

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

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

`a50` A50
Type: estimable
Default: No Default

`ato95` Ato95
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

9. Estimation command and subcommand syntax

9.1. Estimation methods

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

`label` The label of the estimate
Type: string
Default: ""

`type` The prior type for the estimate
Type: string
Default: No Default

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

`lower_bound` The lower bound for the parameter
Type: constant
Default: No Default

`upper_bound` The upper bound for the parameter
Type: constant
Default: No Default

`estimation_phase` The first estimation phase to allow this to be estimated
Type: non-negative integer
Default: 1

`mcmc` Indicates if this parameter is fixed at the point estimate during an MCMC run
Type: boolean
Default: false

`transformation` Type of simple transformation to apply to estimate
Type: string
Default: ""

`transform_with_jacobian` Apply jacobian during transformation
Type: boolean
Default: false

`prior_applies_to_transform` Does the prior apply to the transformed parameter? a legacy switch, see Manual for more information
Type: boolean
Default: false

9.1.1. `@estimate[label].type=beta`

`mu` Beta prior mean (mu) parameter
Type: estimable
Default: No Default

`sigma` Beta prior variance (sigma) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

`a` Beta prior lower bound of the range (A) parameter
Type: constant
Default: No Default

`b` Beta prior upper bound of the range (B) parameter
Type: constant
Default: No Default

9.1.2. `@estimate[label].type=lognormal`

`mu` The lognormal prior mean (mu) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

`cv` The Lognormal variance (CV) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

9.1.3. `@estimate[label].type=normal`

`mu` The normal prior mean (mu) parameter
Type: estimable
Default: No Default

`cv` The normal variance (standard devation) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

9.1.4. `@estimate[label].type=normal_by_stdev`

`mu` The normal prior mean (mu) parameter
Type: estimable
Default: No Default

`sigma` The normal variance (standard devation) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

9.1.5. `@estimate[label].type=normal_log`

`mu` The normal-log prior mean (mu) parameter
Type: estimable
Default: No Default

`sigma` The normal-log prior variance (standard deviation) parameter
Type: estimable
Default: No Default
Lower Bound: 0.0 (exclusive)

9.1.6. `@estimate[label].type=uniform`

9.1.7. `@estimate[label].type=uniform_log`

9.2. Point estimation

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

`label` The minimiser label
Type: string
Default: No Default

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

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

`covariance` Indicates if a covariance matrix should be generated
Type: boolean
Default: true

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

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

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

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

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

9.2.2. `@minimiser[label].type=betadiff`

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

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

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

9.2.3. `@minimiser[label].type=cppad`

`retape` Retape? yes or no
Type: string
Default: ”true”
Allowed Values: true, false

`print_level` Level of debug to stdout
Type: non-negative integer
Default: 5u
Lower Bound: 0u (inclusive)
Upper Bound: 12u (inclusive)

`sb` String buffer output?
Type: string
Default: ”yes”
Allowed Values: yes, no

`max_iter` Maximum number of iterations
Type: non-negative integer
Default: 4000
Lower Bound: 0u (exclusive)

`tol` Tolerance for convergence
Type: constant
Default: 1e-9

`acceptable_tol` Acceptable tolerance
Type: constant
Default: 1e-6
Lower Bound: 0.0 (inclusive)

`acceptable_obj_change_tol`
Type: constant
Default: 1e+20
Lower Bound: 0u (exclusive)

`derivative_test` How to test for derivaties
Type: string
Default: ”none”
Allowed Values: none, first-order, second-order, only-second-order

`point_pertubation_radius`
Type: constant
Default: 0.0
Lower Bound: 0.0 (inclusive)

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

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

`crossover_probability` Define the minimisers crossover probability
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

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

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

9.2.5. **`@minimiser[label].type=d_lib`**

9.2.6. **`@minimiser[label].type=numerical_differences`**

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

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

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

`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 of type *mcmc*

`label` The label of the MCMC
Type: string
Default: No Default

`type` The type of MCMC

Type: string
Default: ””

`length` The number of iterations in for the MCMC chain
Type: non-negative integer
Default: No Default

`active` Indicates if this is the active MCMC algorithm
Type: boolean
Default: true

`print_default_reports` Indicates if the output prints the default reports
Type: boolean
Default: true

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

9.3.1. `@mcmc[label].type=independence_metropolis`

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

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

`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
Type: constant
Default: 0.8

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

`correlation_adjustment_diff` Minimum non-zero variance times the range of the bounds in the covariance matrix of the proposal distribution
Type: constant
Default: 0.0001

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

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

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

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

`adapt_stepsize_method` Method to adapt step size.
Type: string
Default: ratio
Allowed Values: ratio, double_half

9.4. Profiles

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

`label` Label
Type: string
Default: ""

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

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

`upper_bound` The upper bounds
Type: constant
Default: No Default

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

`same` A Parameter that are constrained to have the same value as the parameter being profiled
Type: string
Default: ””

9.5. Defining catchability constants

`@catchability` *`label`* Define an object of type *catchability*

`label` Label of the catchability
Type: string
Default: No Default

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

9.5.1. `@catchability[label].type=free`

`q` value of the catchability
Type: estimable
Default: No Default

9.5.2. `@catchability[label].type=nuisance`

`lower_bound` Upper bound for nuisance catchability
Type: constant
Default: No Default

`upper_bound` Lower bound for nuisance catchability
Type: constant
Default: No Default

9.6. Defining penalties

`@penalty` *`label`* Define an object of type *penalty*

`label` The label of the penalty
Type: string
Default: No Default

`type` The type of penalty
Type: string
Default: No Default

9.6.1. `@penalty[label].type=process`

`multiplier` The penalty multiplier
Type: constant
Default: 1.0

`log_scale` Indicates if the sums of squares is calculated on the log scale
Type: boolean
Default: false

9.7. Defining priors on parameter ratios, differences, and means

`@additional_prior` *`label`* Define an object of type *additional_prior*

`parameter` Name of the parameter to generate additional prior on
Type: string
Default: No Default

`label` Label for teh additional prior
Type: string
Default: No Default

`type` Type of additional prior
Type: string
Default: No Default

9.7.1. `@additional__prior[label].type=beta`

`mu` Beta distribution mean (mu) parameter
Type: constant
Default: No Default

`sigma` Beta distribution variance (sigma) parameter
Type: constant
Default: No Default
Lower Bound: 0.0 (inclusive)

`a` Beta distribution lower bound of the range (A) parameter
Type: constant
Default: No Default

`b` Beta distribution upper bound of the range (B) parameter
Type: constant
Default: No Default

9.7.2. `@additional_prior[label].type=element_difference`

`second_parameter` Name of the second parameter for comparing
Type: string
Default: No Default

`multiplier` Multiply the penalty by this factor
Type: constant
Default: 1

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

`mu` The lognormal prior mean (mu) parameter
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

`cv` The Lognormal variance (CV) parameter
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

9.7.4. `@additional_prior[label].type=uniform_log`

9.7.5. `@additional_prior[label].type=vector_average`

`method` What calculation method to use, either k, l, or m
Type: string
Default: k

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

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

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

`log_scale` Should sums of squares be calculated on the log scale?
Type: boolean
Default: false

`multiplier` Multiply the penalty by this factor
Type: constant
Default: 1

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

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

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

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 of type *observation*

`label` Label
Type: string
Default: No Default

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

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

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

`delta` Robustification value (delta) for the likelihood
Type: constant
Default: DELTA

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

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

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

10.1.1. `@observation[label].type=process_abundance`

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`catchability` The label of the catchability (q) block assumed for this observation
Type: string
Default: No Default

`obs` The observed values
Type: string vector
Default: No Default

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`error_value` The error values of the observed values (note the units depend on the likelihood
Type: constant vector
Default: No Default

`process_error` The label of the process error
Type: constant
Default: 0.0

`process` The label of the process for the observation
Type: string
Default: No Default

`process_proportion` Proportion through the process when the observation is evaluated
Type: constant
Default: 0.5

10.1.2. `@observation[label].type=abundance`

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`catchability` The label of the catchability coefficient (q) assumed
Type: string
Default: No Default

`obs` The observed values
Type: string vector
Default: No Default

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`error_value` The error values of the observed values (note the units depend on the likelihood
Type: constant vector
Default: No Default

`process_error` The label of the process error block
Type: constant
Default: 0.0

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.1.3. `@observation[label].type=process_biomass`

`catchability` The time-step of the observation
Type: string
Default: No Default

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`obs` The observed values
Type: string vector
Default: No Default

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`error_value` The error values of the observed values (note the units depend on the likelihood
Type: constant vector
Default: No Default

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`process_error` Value for process error
Type: estimable
Default: 0.0

`process` The label of the process for the observation
Type: string
Default: No Default

`process_proportion` Proportion through the process when the observation is evaluated
Type: constant
Default: 0.5

10.1.4. `@observation[label].type=biomass`

`catchability` The time-step of the observation
Type: string
Default: No Default

`time_step` The label of time-step that the observation occurs in

Type: string
Default: No Default

`obs` The observed values
Type: string vector
Default: No Default

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`error_value` The error values of the observed values (note the units depend on the likelihood
Type: constant vector
Default: No Default

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`process_error` Value for process error
Type: estimable
Default: 0.0

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.1.5. `@observation[label].type=process_removals_by_age`

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

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

`age_plus` Use age plus group
Type: boolean
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string vector
Default: No Default

`tolerance` Tolerance
Type: constant
Default: 0.001

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` Label of process error to use
Type: estimable vector
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ””

`method_of_removal` Label of observed method of removals
Type: string vector
Default: ””

`mortality_instantaneous_process` The label of the mortality instantaneous process for the observation
Type: string
Default: No Default

10.1.6. `@observation[label].type=process_removals_by_length`

`length_bins` Length bins
Type: constant vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: 0.001

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` the value of process error
Type: constant vector
Default: true

`method_of_removal` Label of observed method of removals
Type: string
Default: ””

`mortality_instantaneous_process` The label of the mortality instantaneous process for the observation
Type: string
Default: No Default

10.1.7. `@observation[label].type=process_proportions_at_age`

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

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

`age_plus` Use age plus group
Type: boolean
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`tolerance` Tolerance on the constraint, that for each year the sum of proportions in each age must equal one e.g. tolerance = 0.1 then 1 - Sum(Proportions) can be as great as 0.1
Type: constant
Default: 0.001

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`selectivities` Labels of the selectivities

Type: string vector
Default: true

`process_errors` Process error
Type: constant vector
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ””

`process` The label of the process for the observation
Type: string
Default: No Default

`process_proportion` Proportion through the process when the observation is evaluated
Type: constant
Default: 0.5

10.1.8. `@observation[label].type=proportions_at_age`

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

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

`age_plus` Use age plus group
Type: boolean
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`tolerance` Tolerance on the constraint, that for each year the sum of proportions in each age must equal one e.g. tolerance = 0.1 then 1 - Sum(Proportions) can be as great as 0.1
Type: constant
Default: 0.001

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`process_errors` Process error
Type: constant vector
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ””

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.1.9. `@observation[label].type=process_proportions_at_length`

`length_bins` Length bins
Type: constant vector
Default: No Default

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: 0.001

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`selectivities` The labels of the selectivities
Type: string vector
Default: true

`process_errors` Process error
Type: constant vector
Default: true

`process` The label of the process for the observation
Type: string
Default: No Default

`process_proportion` Proportion through the process when the observation is evaluated
Type: constant
Default: 0.5

10.1.10. `@observation[label].type=proportions_at_length`

`length_bins` Length bins
Type: constant vector
Default: No Default

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`length_plus_group` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: 0.001

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`selectivities` The labels of the selectivities
Type: string vector
Default: true

`process_errors` Process error
Type: constant vector
Default: true

`time_step_proportion` Proportion through the mortality block of the time step when the

observation is evaluated
Type: constant
Default: 0.5

10.1.11. `@observation[label].type=process_proportions_by_category`

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

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

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`age_plus` Use age plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`selectivities` The labels of the selectivities
Type: string vector
Default: true

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`process` The label of the process for the observation
Type: string
Default: No Default

`process_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.1.12. `@observation[label].type=proportions_by_category`

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

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

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`age_plus` Use age plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`selectivities` The labels of the selectivities
Type: string vector
Default: true

`categories2` Target Categories
Type: string vector
Default: No Default

`selectivities2` Target Selectivities
Type: string vector
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`time_step_proportion` Proportion through the time step to analyse the partition from
Type: constant
Default: 0.5

10.1.13. `@observation[label].type=proportions_migrating`

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

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

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`age_plus` Use age plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`ageing_error` Label of ageing error to use
Type: string
Default: ""

`process` Process label
Type: string
Default: No Default

10.1.14. `@observation[label].type=tag_recapture_by_age`

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

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

`age_plus` Use age plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`categories2` The available categories in the partition
Type: string vector
Default: No Default

`selectivities` The labels of the selectivities
Type: string vector
Default: true

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`selectivities2` The categories of tagged individuals for the observation
Type: string vector
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`detection` Probability of detecting a recaptured individual
Type: constant
Default: No Default

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.1.15. `@observation[label].type=tag_recapture_by_length`

`length_bins` Length Bins
Type: constant vector
Default: No Default

`plus_group` Indicates if the last length bin is a plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`categories2` The categories of tagged individuals for the observation
Type: string vector
Default: No Default

`selectivities` The labels of the selectivities used for untagged categories
Type: string vector
Default: true

`selectivities2` The labels of the tag category selectivities
Type: string vector
Default: No Default

`process_errors` Process error
Type: constant vector
Default: true

`detection` Probability of detecting a recaptured individual
Type: constant
Default: No Default

`dispersion` Over-dispersion parameter (phi)
Type: constant
Default: 1.0

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: 0.5

10.2. Likelihoods

`@likelihood` *`label`* Define an object of type *likelihood*

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

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

- None
- Data
- 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 of type *ageing_error*

`label` Label of the ageing error
Type: string
Default: No Default

`type` Type of ageing error
Type: string
Default: No Default

10.3.1. `@ageing_error[label].type=data`

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

10.3.3. `@ageing_error[label].type=normal`

`cv` CV of the misclassification matrix
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)

`k` k defines the minimum age of individuals which can be misclassified, e.g., individuals of age less than k have no ageing error
Type: non-negative integer
Default: 0u

10.3.4. `@ageing_error[label].type=off_by_one`

`p1` proportion misclassified as one year younger, e.g., the proportion of age 3 individuals that were misclassified as age 2
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`p2` proportion misclassified as one year older, e.g., the proportion of age 3 individuals that were misclassified as age 4
Type: estimable
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`k` The minimum age of fish which can be misclassified, i.e., fish of age less than k are assumed to be correctly classified
Type: non-negative integer
Default: 0u
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 of type *report*

`label` The label for the report

Type: string
Default: No Default

`type` The type of report
Type: string
Default: No Default

`file_name` The File Name if you want this report to be in a seperate file
Type: string
Default: ””

`write_mode` The write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.1. `@report[label].type=addressable`

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

`years` Years to print the addressable for
Type: non-negative integer vector
Default: No Default

`time_step` Time Step label
Type: string
Default: ””

11.1.2. `@report[label].type=ageing_error_matrix`

`ageing_error` Ageing Error label
Type: string
Default: No Default

11.1.3. `@report[label].type=catchabilities`

11.1.4. `@report[label].type=category_info`

11.1.5. `@report[label].type=category_list`

11.1.6. `@report[label].type=correlation_matrix`

11.1.7. `@report[label].type=covariance_matrix`

11.1.8. `@report[label].type=derived_quantity`

11.1.9. `@report[label].type=equation_test`

`equation` Equation to do a test run of
Type: string vector
Default: No Default

11.1.10. `@report[label].type=estimate_summary`

11.1.11. `@report[label].type=estimate_value`

11.1.12. `@report[label].type=hessian_matrix`

11.1.13. `@report[label].type=initialisation_partition`

11.1.14. `@report[label].type=mcmc_covariance`

11.1.15. `@report[label].type=mcmc_objective`

11.1.16. `@report[label].type=mcmc_sample`

11.1.17. `@report[label].type=m_p_d`

11.1.18. `@report[label].type=objective_function`

11.1.19. `@report[label].type=observation`

`observation` Observation label
Type: string
Default: No Default

`pearsons_residuals` Print Pearsons Residuals
Type: boolean
Default: false

`normalised_residuals` Print Normalised Residuals

Type: boolean
Default: false

11.1.20. `@report[label].type=output_parameters`

11.1.21. `@report[label].type=partition`

`time_step` Time Step label
Type: string
Default: ””

`years` Years
Type: non-negative integer vector
Default: true

11.1.22. `@report[label].type=partition_biomass`

`time_step` Time Step label
Type: string
Default: ””

`years` Years
Type: non-negative integer vector
Default: true

`units` Units (Default Kgs
Type: string
Default: kgs

11.1.23. `@report[label].type=partition_mean_weight`

`time_step` Time Step label
Type: string
Default: ””

`years` Years
Type: non-negative integer vector
Default: true

11.1.24. `@report[label].type=process`

`process` Process label that is reported
Type: string
Default: ””

11.1.25. `@report[label].type=project`

`project` Project label that is reported
Type: string
Default: ””

11.1.26. `@report[label].type=random_number_seed`

11.1.27. `@report[label].type=selectivity`

`selectivity` Selectivity name
Type: string
Default: No Default

11.1.28. `@report[label].type=simulated_observation`

`observation` Observation label
Type: string
Default: No Default

11.1.29. `@report[label].type=standard_header`

11.1.30. `@report[label].type=time_varying`

12. Including commands from other files

`@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.csl2"`
Note: `@include` does not denote the end of the previous command block as is the case for all other commands

13. Syntax conventions, examples and niceties

13.1. Input File Specification

The file format used for CASAL2 is based on the formats used for CASAL and SPM. It's a standard text file that contains definitions organised into blocks.
Without exception, every object specified in a configuration file is part of a block. At the top level blocks have a one-to-one relationships with components in the system.
Example:

```
@block1 label
parameter value
parameter value_1 value 2

@block2 label
parameter value
table table_name
column_1 column_2
data_1 data_2
data_3 data_4
end_table
```

Some general notes about writing configuration files:

1. Whitespace can be used freely. Tabs and spaces are both accepted
2. A block ends only at the beginning of a new block or end of final configuration file
3. You can include another configuration file from anywhere
4. Included files are placed inline, so you can continue a block in a new file
5. The configuration files support inline declarations of objects

13.1.1. Keywords And Reserved Characters

In order to allow efficient creation of input files CASAL2's file format contains special keywords and characters that cannot be used for labels etc.

`@Block` Definitions

Every new block in the configuration file must start with a block definition character. The reserved character for this is the `@`character
Example:

```
@block1 <label>
type <type>

@block2 <label>
type <type>
```

'type' Keyword

The 'type' keyword is used for declaring the sub-type of a defined block. Any block object that has multiple sub-types will use the type keyword.
Example:

```
@block1 <label>
type <sub_type>

@block2 <label>
type <sub_type>
```

# (Single-Line Comment)

Comments are supported in the configuration file in either single-line (to end-of-line) or multi-line
Example:

```
@block <label>
type <sub_type> #Descriptive comment
#parameter <value_1>  This whole line is commented out
parameter <value_1> #<value_2>(value_2 is commented out)
```

/* */ (Multi-Line Comment)

Multiple line comments are supported by surrounding the comments in /* and */
Example:

```
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>

\*
Do not load this process
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>
*\
```

{} (Indexing Parameters)

Users can reference individual elements of a map using the { } syntax, for example when estimating `ycs_values` you may only want to estimate a block of YCS not all of them say between 1975 and 2012. Example:

```
@estimate YCS
parameter process[Recruitment].ycs_values{1975:2012}
type uniform
lower_bound
upper_bound
```

':' (Range Specifier)

The range specifier allows you to specify a range of values at once instead of having to input them manually. Ranges can be either incremental or decremental.
Example:

```
@process my_recruitment_process
type constant_recruitment
```

```
years_to_run 1999:2009 #With range specifier

@process my_mortality_process
type natural_mortality
years_to_run 2000 2001 2002 2003 2004 2005 2006 2007 #Without range specifier
```

',' (List Specifier)

When a parameter supports multiple values in a single entry you can use the list specifier to supply multiple values as a single parameter.
Example:

```
@categories
format sex.stage
names male,female.immature,mature #With list specifier

@categories
format sex.stage
names male.immature male.mature female.immature female.mature #Without list specifier
```

'table' and 'end_table' Keyword

The table keyword is used to define a table of information used as a parameter. The line following the table declaration must contain a list of columns to be used. Following lines are rows of the table. Each row must have the same number of values as the number of columns specified. The table definition must end with the 'end_table' keyword on it's own line. The first row of a table will be the name of the columns if required.
Example:

```
@block <label>
type <sub_type>
parameter <value_1>
table <table_label>
<column_1> <column_2> <column_n>
<row1_value1> <row1_value2> <row1_valueN>
<row2_value1> <row2_value2> <row2_valueN>
end_table
```

[ ] (Inline Declarations)

When an object takes the label of a target object as a parameter this can be replaced with an inline declaration. An inline declaration is a complete declaration of an object one 1 line. This is designed to allow the configuration writer to simplify the configuration writing process.
Example:

```
#With inline declaration with label specified for time step
@model
time_steps step_one=[type=iterative; processes=recruitment ageing]

#With inline declaration with default label (model.1)
@model
time_steps [type=iterative; processes=recruitment ageing]

#Without inline declaration
@model
```

```
time_steps step_one

@time_step step_one
processes recruitment ageing
```

Categories

The CASAL2 model is essentially a 2-dimensional model. The model partition is:
Categories x Ages/Lengths.

Each category supports the ability to have a different range of ages/lengths and accessibility during different time periods.

Because each category is quite complicated the syntax for defining categories has been structured to allow complex definitions using a simple short-hand structure.

The ”format” parameter allows you to tell the model the structure of the category labels. By using a ”.” (period) character between each segment we can utilise this later in the model to do short-hand lookups of categories.

The ”names” parameter is a list of the category names. The syntax of these names will need to match the ”format” parameter so CASAL2 can organise and search on them. Using the ”list specifier” and range characters we can shorten this parameter significantly.
Example:

```
@categories
format sex.stage.tag
names male.immature.notag male.immature.2001 male.mature.notag male.mature.2001

names male.immature #Invalid: No tag information
names female #Invalid: no stage of tag information
names female.immature.notag.1 #Invalid: Extra format segment not defined

names male,female.immature,mature.notag,2001:2005 #OK!
#Without short-hand. You'd have to write:
names male.immature.notag male.immature.2001 male.immature.2002 male.immature.2003 male.immature.20
```

When we have specific data for a year in a category we don’t want the model to process this category during other years (or the initialisation stages). We can define a list of years where each category will be available, this will override the default of all years in the model. Any category where you overwrite the default will no longer be accessible in the initialisation phases.
Examples:

```
@model
start_year 1998
final_year 2010

@categories
format sex.stage.tag
names male,female.immature,mature.notag,2001:2005 #OK!
years tag=2001=1999:2003  tag=2005=2003:2007
# Categories with the tag value 2001 will be available during years 1999, 2000, 2001, 2002 and 2003
# Categories with the tag value 2005 will be available during the years 2003, 2004, 2005, 2006, 200
```

13.2. More examples of shorthand syntax and use of CASAL2's reserved and key characters

Categories

CASAL2 allows many user defined categories so shorthand syntax has been added to aid in the readability of complex configuration scripts and partition structures. For example when defining categories you can use a comma for shortening lists of categories. The following syntax is how we would specify the categories the long way.

```
@categories
format sex.stage
names male.immature male.mature female.immature female.mature
```

for the exact same partition structure but specified in a shorter way users could define the categories as, (note the use of the list character ','),

```
@categories
format sex.stage
names male,female.immature,mature
```

CASAL2 asks for categories in processes and observations so that it can apply the right model dynamics to the right elements of the partition. For the same reason as defining categories shorthand syntax aids in readability and input management. An example of a process where categories need to be supplied as an input command is in ageing,

```
# 1. The standard way
@ageing my_ageing
categories male.immature male.mature female.immature female.mature

# 2. The 1st short-hand way
@ageing my_ageing
categories male,female.immature,mature

# 3. Wild Card (all categories)
@ageing my_ageing
categories *

# 4. The 2nd short-hand way
@ageing my_ageing
categories sex=male sex=female
```

Sometimes in observations we want to amalgamate categories together for example if we had a biomass estimate of the population that was made up of both males and females in the population you can specify this using the + special character, for example

```
@observation CPUE
type biomass
catchability Fishq
time_step one
categories male+female
selectivities FishSel
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
```

Shorthand syntax can be useful when applying processes to a select group of categories from the partition, for example. If we wanted to apply a spawning migration to the mature categories in the partition and the partition was defined by the categories below,

```
@categories
format area.maturity.tag
names north.immature.notag,2011 north.mature.notag,2011 south.immature.notag,2011
south.mature.notag,2011
```

If we wanted to migrate a portion of the mature population from the southern area to the northern are you could use the following syntax,

```
@process spawn_migration
type transition_category
from format=south.mature.*
to format=north.mature.*
proportions 1.0
selectivities One
```

Parameters

CASAL2 also allows parameters that are of type vector or map to be referenced and estimated partially. An example of a parameter that is type vector is `ycs_values` in a recruitment process. Let say a recruitment block was specified as follows,

```
@process WestRecruitment
type recruitment_beverton_holt
r0 400000
years
ycs_values 1 1 1 1 1 1 1 1
ycs_years 1975:1983
An alternative specification to the sequence of values you can use an astrix to
shorthand repeating integers e.g.

ycs_values 1*8

steepness 0.9
age 1
```

Lets say we wanted to only estimate the last four years of the parameter `process[WestRecruitment].ycs_values`. This can be done as specified in the following `@estimate` block,

```
@estimate
parameter process[WestRecruitment].ycs_values{1979:1983}
type uniform
lower_bound 0.1 0.1 0.1 0.1
upper_bound  10  10  10  10
```

Note the first element of a vector is indexed by 1. This syntax can be applied to parameters that are of type map as well, for information on what type a parameter is see the syntax section. An example of a parameter that is of type map is `@time_varying[label].type=constant`. For the following `@time_varying` block,

```
@time_varying q_step1
type constant
parameter catchability[Fishq].q
years  1992 1993 1994 1995
value  0.2 0.2 0.2 0.2
```

In this example a user may want to estimate only one element of the map (say 1992), but force all other years to be the same as the one estimate. This can be done in an estimate block as follows,

```
@estimate
parameter time_varying[q_step1].value{1992}
same time_varying[q_step1].value{1993:1995}
type uniform
lower_bound 0.1 0.1 0.1 0.1
upper_bound  10  10  10  10
```

In line declaration

In line declarations can help shorten models 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 catchability without explicitly 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^th selectivity would be labelled;

```
@selectivity halfm.10
```

13.3. Processes

Processes are special in how they can be defined, althrough out this document we have been referring to specifying a process as follows,

```
@process Recruitment
type recruitment_beverton_holt
```

However for convenience and for file clarity you could equally specify this block as follows,

```
@recruitment Recruitment
type beverton_holt
```

The trick is that you can replace the keyword `process` with the first word of the process type, in the example above this is the `recruitment` this can be away of creating more reader friendly/lay term configuration scripts. More examples follow;

```
@mortality Fishing_and_M
type instantaneous


@transition Migration
type category
```

13.4. 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.csl2`. In this example, `casal2.csl2` 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,

```
#THE MODEL
@model
start_year 1975
final_year 2012
min_age 1
max_age 30
age_plus true
base_weight_units tonnes
initialisation_phases Equilibrium_state
time_steps Sep_Feb Mar_May Jun_Aug

#CATEGORIES
#
@categories
format stock ## Single sex category
names HAK4
age_lengths age_size

@initialisation_phase Equilibrium_state
type derived

## Define the Annual Cycle
## This is a list of labels that correspond to a process
@time_step Sep_Feb
processes Recruitment Instantaneou_Mortality

@time_step Mar_May
processes Instantaneou_Mortality
```

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.

14. Post processing output using R

In the downloaded bundle is a R-package that reads CASAL2 output into **R**. The CASAL2 package has multiple functionalities, it can dump CASAL2 into **R** as a list. It also has plotting and summarising functions for summarising CASAL2 output.

The `extract.mpd()` function will read and process CASAL2 output files into **R** that have been generated from the following CASAL2 runs, `-r`, `-e`, `-p`, `-f` that has not been run with the `--tabular`. This function expects a certain format of the output (i.e every report starts with * and finishs with *end), if this format is broken the library will most likely spit the dummy. CASAL2's reported output is written so that each `@report` will start with a '*' and end with '*end'. Users can use this format as the basis to construct **R** or other functions that read CASAL2 output to identify and read individual reports for post-processing.

The `extract.tabular()` function will read and process CASAL2 output files into **R** that have been generated from any CASAL2 run that has been run with the `--tabular`. Although the

The `extract.mcmc()` function will read and process CASAL2 samples and objectives files into **R** that have been generated from CASAL2 `-m` runs.

These extract functions differ by how the expected output is structured and they each create a different `casal2` object in **R** that when run with `summary, plot` functions will generate different plots. The following is a list of R functions with their descriptions that are available in the R package.

- `extract.mpd()` - import CASAL2 default output into **R** as a list
- `extract.mcmc()` - import CASAL2 mcmc output into **R** as a list
- `extract.tabular()` - import CASAL2 tabular output into **R** as a list
- `extract.parameters()` - import CASAL2 parameter files into **R** as a list
- `plot.derived_quantities()` - plot derived quantities
- `plot.selectivities()` - plot selectivities
- `summary()` - summarise a model run
- `extract.csl2.file` - read CASAL2 .csl2 (configuration) file into **R** as a list
- `write.csl2.file` - write CASAL2 .csl2 (configuration) file from R
- `ReadSimulatedData` - read CASAL2 output into **R** from a `casal2 -s` run
- `param.profile()` - plot a parameter profile, from a `casal2 -p` run
- `param.profile.by.cohort` - plot a parameter profile for an age compositional dataset by cohort and category, from a `casal2 -p` run

specific settings for these functions can be queried once you have loaded the CASAL2 R library into your current R session via (`library(CASAL2)`), using the standard **R** help syntax `?` (i.e. `?param.profile()`)

Standard diagnostics and plots for model output

An aspect of CASAL2 that we have added to aid in userability is an **R** package that has functions that can summarise and diagnose a model with ease. Stock synthesis is exceptional for having a well tested and used **R** package, this our attempt of trying to follow suit.

`plot.mpd.derived.quantity()`

`plot.mpd.derived.quantity()` Generally when comparing model outputs we are either comparing different parameters for the same model structure (Option 1) or comparing outputs from multiple model structures (Option 2). This function in theory should work for both of these.

`plot.mpd.selectivities()`

`plot.mcmc.pearsons.residuals()`

DataWeighting

An important component of stock assessment is dealing with data conflicts through the use of dataweighting. There are a range of methods for this for a great summary see **Reference Chris's papers.**. We have re-written Chris's DataWeighting

15. Troubleshooting

15.1. Introduction

This section is to aid users in debugging models, if you cannot resolve an issue using these guidelines then don't hesitate to contact the development team. To report an issue please follow the format described in Section 15.3. We are hoping that most user errors will be well documented and that CASAL2 will produce informative error messages. In the case where this doesn't happen, there are some quick and easy tactics that users can do to attempt to resolve or at least isolate an error/bug. Using CASAL2's internal logging out system, this is invoked at the command line with by the `--loglevel` parameter followed by one of these arguments; `trace`, `finest`, `fine`, `medium`. An example of implementing logging with trace level at the command line is,

```
casal2 -r --loglevel trace > output.log 2> log.out
```

The above command will output CASAL2 normal reports into the file "output.log" where as the `2>` syntax will print the error logged out information into the file "log.out". You should be able to see where CASAL2 is exiting by going to the end of the "log.out" file.

Deciding what argument to get CASAL2 to log out at. This will depend on what mode you are running CASAL2 and at what level information you want to see. There is a hierarchy in the arguments with `medium` being the most coarse, and `trace` being the finest mode with `fine` and `finest` in between. We suggest that if you are running CASAL2 in an iterative state such as estimation (`casal2 -e`) or mcmc you only use `medium`. This is because the logging out can print a great deal of information from a single model run, so naturally when you are doing an estimation which could equate to thousands of model runs some of the finer log outs can produce gigabyte sized text files that cannot be opened. For a single iteration run such as `casal2 -r` all the logging out arguments will be useful. If you would like to see how CASAL2 creates these reports you can open up almost any ".cpp" file and see code lines such as in `model.cpp`,

```
LOG_FINE() << "Model:  State change to Execute";
```

taken from line 379.

15.2. Reporting errors

If you find a bug or problem in CASAL2, please let the Development Team know by contacting them at casal2@niwa.co.nz or by submitting an issue on the github repository found at `https://github.com/NIWAFisheriesModelling/CASAL2/issues`. The latter is preferred as it will automatically document the issue which is better than depending on the development team, who may be forgetful. Please follow the guidelines below, as they will enhance the debugging process which can be quite time consuming.

15.3. Guidelines for reporting a problem with CASAL2

1. Check to ensure you are using the most recent version of CASAL2. Its possible that the error or problem you are having may have ready been resolved.
2. Describe the version of CASAL2 are you using? e.g., CASAL2 v2017-07-03 (rev. 4480b20e) Microsoft Windows executable". The version is provided by CASAL2 with the following command `casal2 -v`.

3. What operating system or environment are you using? e.g., “IBM-PC Intel CPU running Microsoft Windows 10 Enterprise”.
4. Give a brief one-line description of the problem, e.g., “a segmentation fault was reported”.
5. 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.”
6. 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.”
7. If the problem causes any error messages to appear, please give the *exact* text displayed, e.g., `segmentation fault (core dumped)`.
8. Remember to attach all relevant input and output files so that the problem can be reproduced (it can be helpful to compress these into a single file e.g. zip 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. CASAL2 software license

GNU GENERAL PUBLIC LICENSE

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".

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

17. Acknowledgements

We thank the developers of CASAL (Bull et al., 2012) for their ideas that led to the development of the CASAL2. The CASAL2 logo was designed by Ian Doonan and Erika Mackay (NIWA).

Much of the structure of CASAL2, equations, and documentation in this manual draw heavily on similar components of the fisheries population model CASAL (Bull et al., 2012) and the spatial model SPM (Dunn et al., 2015). We thank the authors of CASAL and SPM for their permission to use their work as the basis for parts of CASAL2 and allow the use of the definitions, concepts, and documentation.

The development of CASAL2 was funded by the New Zealand Ministry for Primary Industries and the National Institute of Water & Atmospheric Research Ltd. (NIWA) under NIWAs Fisheries Centre Research Programme 1.

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, National Institute of Water and Atmospheric Research Ltd (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, S. Rasmussen, and S. Mormede. Spatial population model user manual, spm v1.1-2016-03-04 (rev. 1278). Technical Report 138, National Institute of Water and Atmospheric Research Ltd (NIWA), 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.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

C S Holling. The components of predation as revealed by a study of small-mammal predation of the european pine sawfly. *The Canadian Entomologist*, 91:293–320, 1959.

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.

M I Michaelis and L Menten. Die kinetik der invertinwirkung. *Biochemische Zeitschrift*, 49:333369, 1913.

Andre E. Punt and Ray Hilborn. *BAYES-SA. Bayesian stock assessment methods in fisheries. User's manual. FAO Computerized information series (fisheries) 12*. Food and Agriculture Organisation of the United Nations, Rome (Italy), 2001.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`.

J Schnute. A versatile growth model with statistically stable parameters. *Canadian Journal of Fisheries and Aquatic Sciences*, 38(9):1128–1140, 1981.

C Sherlock and G Roberts. Optimal scaling of the random walk metropolis on elliptically symmetric unimodal targets. *Bernoulli*, 15(3), 2009.

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.

C.J Walters and D Ludwig. Calculation of bayes posterior probability distributions for key population parameters. *Canadian Journal of Fisheries and Aquatic Sciences*, 1994.

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. Index

20. Quick reference

@ageing_error *label* Define an object of type *ageing_error*

`label` Label of the ageing error
`type` Type of ageing error

@ageing__error[label].type=data

@ageing__error[label].type=none

@ageing__error[label].type=normal

`cv` CV of the misclassification matrix
`k` k defines the minimum age of individuals which can be misclassified, e.g., individuals of age less than k have no ageing error

@ageing__error[label].type=off_by_one

`p1` proportion misclassified as one year younger, e.g., the proportion of age 3 individuals that were misclassified as age 2
`p2` proportion misclassified as one year older, e.g., the proportion of age 3 individuals that were misclassified as age 4
`k` The minimum age of fish which can be misclassified, i.e., fish of age less than k are assumed to be correctly classified
@age_length *label* Define an object of type *age_length*

`label` Label of the age length relationship
`type` Type of age length relationship
`time_step_proportions` the fraction of the year applied in each time step that is added to the age for the purposes of evaluating the length, i.e., a value of 0.5 for a time step will evaluate the length of individuals at age+0.5 in that time step
`distribution` The assumed distribution for the growth curve
`cv_first` CV for the first age class
`cv_last` CV for last age class
`casal_switch` If true, use the (less accurate) equation for the cumulative normal function as was used in the legacy version of CASAL.

@age__length[label].type=data

`external_gaps`
`internal_gaps`
`length_weight` The label from an associated length-weight block
`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age

@age__length[label].type=none

@age__length[label].type=schnute

`y1` Define the y1 parameter of the Schnute relationship
`y2` Define the y2 parameter of the Schnute relationship
`tau1` Define the τ_1 parameter of the Schnute relationship
`tau2` Define the τ_2 parameter of the Schnute relationship
`a` Define the *a* parameter of the Schnute relationship
`b` Define the *b* parameter of the Schnute relationship
`length_weight` Define the label of the associated length-weight relationship
`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age

@age__length[label].type=von_bertalanffy

`linf` Define the $L_{infinity}$ parameter of the von Bertalanffy relationship
`k` Define the k parameter of the von Bertalanffy relationship
`t0` Define the t_0 parameter of the von Bertalanffy relationship
`length_weight` Define the label of the associated length-weight relationship
`by_length` Specifies if the linear interpolation of CV's is a linear function of mean length at age. Default is just by age
@catchability *label* Define an object of type *catchability*

`label` Label of the catchability
`type` Type of catchability

@catchability[label].type=free

`q` value of the catchability

@catchability[label].type=nuisance

`lower_bound` Upper bound for nuisance catchability
`upper_bound` Lower bound for nuisance catchability
@derived_quantity *label* Define an object of type *derived_quantity*

`label` Label of the derived quantity
`type` Type of derived quantity
`time_step` The time step in which to calculate the derived quantity after
`categories` The list of categories to use when calculating the derived quantity
`selectivities` A list of one selectivity
`time_step_proportion` Proportion through the mortality block of the time step when calculated
`time_step_proportion_method` Method for interpolating for the proportion through the mortality block

@derived__quantity[label].type=abundance

@derived__quantity[label].type=biomass

@estimate *label* Define an object of type *estimate*

`label` The label of the estimate
`type` The prior type for the estimate
`parameter` The name of the parameter to estimate in the model
`lower_bound` The lower bound for the parameter
`upper_bound` The upper bound for the parameter
`estimation_phase` The first estimation phase to allow this to be estimated
`mcmc` Indicates if this parameter is fixed at the point estimate during an MCMC run
`transformation` Type of simple transformation to apply to estimate
`transform_with_jacobian` Apply jacobian during transformation
`prior_applies_to_transform` Does the prior apply to the transformed parameter? a legacy switch, see Manual for more information

@estimate[label].type=beta

`mu` Beta prior mean (mu) parameter
`sigma` Beta prior variance (sigma) parameter
`a` Beta prior lower bound of the range (A) parameter
`b` Beta prior upper bound of the range (B) parameter

@estimate[label].type=lognormal

`mu` The lognormal prior mean (mu) parameter
`cv` The Lognormal variance (CV) parameter

`@estimate[label].type=normal`

`mu` The normal prior mean (mu) parameter
`cv` The normal variance (standard devation) parameter

`@estimate[label].type=normal_by_stdev`

`mu` The normal prior mean (mu) parameter
`sigma` The normal variance (standard devation) parameter

`@estimate[label].type=normal_log`

`mu` The normal-log prior mean (mu) parameter
`sigma` The normal-log prior variance (standard deviation) parameter

`@estimate[label].type=uniform`

`@estimate[label].type=uniform_log`

`@estimate_transformation` *`label`* Define an object of type *estimate_transformation*

`label` Label for the transformation block
`type` Type of transformation
`estimate_label` Label of estimate block to apply transformation to
`transform_with_jacobian` Apply jacobian during transformation

`@estimate__transformation[label].type=average_difference`

`difference_estimate_label` The label from the @estimate block relating to the difference parameter, used in this transformation

`@estimate__transformation[label].type=inverse`

`@estimate__transformation[label].type=log`

`@estimate__transformation[label].type=log_sum`

`second_estimate` The label from the @estimate block relating to the difference parameter, used in this transformation

`@estimate__transformation[label].type=orthogonal`

`@estimate__transformation[label].type=simplex`

`@estimate__transformation[label].type=square_root`

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

`label` The label of the initialisation phase
`type` The type of initialisation

`@initialisation__phase[label].type=cinitial`

`categories` The list of categories for the Cinitial initialisation

`@initialisation__phase[label].type=derived`

`insert_processes` Additional processes not defined in the annual cycle, that are to beinserted into this

initialisation phase
`exclude_processes` Processes in the annual cycle to be excluded from this initialisation phase
`casal_intialisation_switch` Run an extra annual cycle to evalaute equilibrium SSB's. Warning - if true, this may not correctly evaluate the equilibrium state. Use true if attempting to replicate a legacy CASAL model

`@initialisation__phase[label].type=iterative`

`years` The number of iterations (years) over which to execute this initialisation phase
`insert_processes` (years) over which to execute this initialisation phase
`exclude_processes` Processes in the annual cycle to be excluded from this initialisation phase
`convergence_years` The iteration (year) when the test for converegence (lambda) is evaluated
`lambda` The maximum value of the absolute sum of differences (lambda) between the partition at year-1 and year that indicates successfull convergence

`@initialisation__phase[label].type=state_category_by_age`

`categories` The list of categories for the category state initialisation
`min_age` The minimum age of values supplied in the definition of the category state
`max_age` The minimum age of values supplied in the definition of the category state
`@likelihood` *`label`* Define an object of type *likelihood*

`@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 of type *derived_quantity*

`label` Label of the derived quantity
`type` Type of derived quantity
`time_step` The time step in which to calculate the derived quantity after
`categories` The list of categories to use when calculating the derived quantity
`selectivities` A list of one selectivity
`time_step_proportion` Proportion through the mortality block of the time step when calculated
`time_step_proportion_method` Method for interpolating for the proportion through the mortality block

`@derived__quantity[label].type=abundance`

`@derived__quantity[label].type=biomass`

`@mcmc` *`label`* Define an object of type *mcmc*

`label` The label of the MCMC
`type` The type of MCMC
`length` The number of iterations in for the MCMC chain
`active` Indicates if this is the active MCMC algorithm
`print_default_reports` Indicates if the output prints the default reports
`step_size` Initial stepsize (as a multiplier of the approximate covariance matrix

`@mcmc[label].type=independence_metropolis`

`start` Covariance multiplier for the starting point of the MCMC
`keep` Spacing between recorded values in the MCMC
`max_correlation` Maximum absolute correlation in the covariance matrix of the proposal distribution
`covariance_adjustment_method` Method for adjusting small variances in the covariance proposal matrix
`correlation_adjustment_diff` Minimum non-zero variance times the range of the bounds in the covariance matrix of the proposal distribution
`proposal_distribution` The shape of the proposal distribution (either the t or the normal distribution
`df` Degrees of freedom of the multivariate t proposal distribution
`adapt_stepsize_at` Iterations in the chain to check and resize the MCMC stepsize
`adapt_covariance_matrix_at` Iterations in the chain to check and resize the MCMC stepsize
`adapt_stepsize_method` Method to adapt step size.
`@minimiser` *`label`* Define an object of type *minimiser*

`label` The minimiser label
`type` The type of minimiser to use
`active` Indicates if this minimiser is active
`covariance` Indicates if a covariance matrix should be generated

`@minimiser[label].type=adolc`

`iterations` Maximum number of iterations
`evaluations` Maximum number of evaluations
`tolerance` Tolerance of the gradient for convergence
`step_size` Minimum Step-size before minimisation fails

`@minimiser[label].type=betadiff`

`iterations` Maximum number of iterations
`evaluations` Maximum number of evaluations
`tolerance` Tolerance of the gradient for convergence

`@minimiser[label].type=cppad`

`retape` Retape? yes or no
`print_level` Level of debug to stdout
`sb` String buffer output?
`max_iter` Maximum number of iterations
`tol` Tolerance for convergence
`acceptable_tol` Acceptable tolerance
`acceptable_obj_change_tol`
`derivative_test` How to test for derivaties
`point_pertubation_radius`

`@minimiser[label].type=de_solver`

`population_size` The number of candidate solutions to have in the population
`crossover_probability` Define the minimisers crossover probability
`difference_scale` The scale to apply to new solutions when comparing candidates
`max_generations` The maximum number of iterations to run
`tolerance` The total variance between the population and best candidate before acceptance
`method` The type of candidate generation method to use

`@minimiser[label].type=d_lib`

`@minimiser[label].type=numerical_differences`

`iterations` Maximum number of iterations

`evaluations` Maximum number of evaluations
`tolerance` Tolerance of the gradient for convergence
`step_size` Minimum Step-size before minimisation fails
`@model` *`label`* Define an object of type *model*

`start_year` Define the first year of the model, immediately following initialisation
`final_year` Define the final year of the model, excluding years in the projection period
`min_age` Minimum age of individuals in the population
`max_age` Maximum age of individuals in the population
`age_plus` Define the oldest age as a plus group
`initialisation_phases` Define the labels of the phases of the initialisation
`time_steps` Define the labels of the time steps, in the order that they are applied, to form the annual cycle
`projection_final_year` Define the final year of the model in projection mode
`length_bins`
`base_weight_units` Define the units for the base weight. This will be the default unit of any weight input parameters
`@observation` *`label`* Define an object of type *observation*

`label` Label
`type` Type of observation
`likelihood` Type of likelihood to use
`categories` Category labels to use
`delta` Robustification value (delta) for the likelihood
`simulation_likelihood` Simulation likelihood to use
`likelihood_multiplier` Likelihood score multiplier
`error_value_multiplier` Error value multiplier for likelihood

`@observation[label].type=process_abundance`

`selectivities` Labels of the selectivities
`time_step` The label of time-step that the observation occurs in
`catchability` The label of the catchability (q) block assumed for this observation
`obs` The observed values
`years` The years of the observed values
`error_value` The error values of the observed values (note the units depend on the likelihood
`process_error` The label of the process error
`process` The label of the process for the observation
`process_proportion` Proportion through the process when the observation is evaluated

`@observation[label].type=abundance`

`selectivities` Labels of the selectivities
`time_step` The label of time-step that the observation occurs in
`catchability` The label of the catchability coefficient (q) assumed
`obs` The observed values
`years` Years for which there are observations
`error_value` The error values of the observed values (note the units depend on the likelihood
`process_error` The label of the process error block
`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated

`@observation[label].type=process_biomass`

`catchability` The time-step of the observation

`time_step` The label of time-step that the observation occurs in
`obs` The observed values
`years` The years of the observed values
`error_value` The error values of the observed values (note the units depend on the likelihood
`selectivities` Labels of the selectivities
`process_error` Value for process error
`process` The label of the process for the observation
`process_proportion` Proportion through the process when the observation is evaluated

@observation[label].type=biomass

`catchability` The time-step of the observation
`time_step` The label of time-step that the observation occurs in
`obs` The observed values
`years` The years of the observed values
`error_value` The error values of the observed values (note the units depend on the likelihood
`selectivities` Labels of the selectivities
`process_error` Value for process error
`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated

@observation[label].type=process_removals_by_age

`min_age` Minimum age
`max_age` Maximum age
`age_plus` Use age plus group
`time_step` The label of time-step that the observation occurs in
`tolerance` Tolerance
`years` Years for which there are observations
`process_errors` Label of process error to use
`ageing_error` Label of ageing error to use
`method_of_removal` Label of observed method of removals
`mortality_instantaneous_process` The label of the mortality instantaneous process for the observation

@observation[label].type=process_removals_by_length

`length_bins` Length bins
`time_step` Time step to execute in
`length_plus_group` Is the last bin a plus group
`tolerance` Tolerance for rescaling proportions
`years` Years for which there are observations
`process_errors` the value of process error
`method_of_removal` Label of observed method of removals
`mortality_instantaneous_process` The label of the mortality instantaneous process for the observation

@observation[label].type=process_proportions_at_age

`min_age` Minimum age
`max_age` Maximum age
`age_plus` Use age plus group
`time_step` The label of time-step that the observation occurs in
`tolerance` Tolerance on the constraint, that for each year the sum of proportions in each age must equal

one e.g. tolerance = 0.1 then 1 - Sum(Proportions) can be as great as 0.1
years The years of the observed values
selectivities Labels of the selectivities
process_errors Process error
ageing_error Label of ageing error to use
process The label of the process for the observation
process_proportion Proportion through the process when the observation is evaluated

@observation[label].type=proportions_at_age

min_age Minimum age
max_age Maximum age
age_plus Use age plus group
time_step The label of time-step that the observation occurs in
tolerance Tolerance on the constraint, that for each year the sum of proportions in each age must equal one e.g. tolerance = 0.1 then 1 - Sum(Proportions) can be as great as 0.1
years The years of the observed values
selectivities Labels of the selectivities
process_errors Process error
ageing_error Label of ageing error to use
time_step_proportion Proportion through the mortality block of the time step when the observation is evaluated

@observation[label].type=process_proportions_at_length

length_bins Length bins
time_step The label of time-step that the observation occurs in
length_plus_group Is the last bin a plus group
tolerance Tolerance for rescaling proportions
years Years for which there are observations
selectivities The labels of the selectivities
process_errors Process error
process The label of the process for the observation
process_proportion Proportion through the process when the observation is evaluated

@observation[label].type=proportions_at_length

length_bins Length bins
time_step The label of time-step that the observation occurs in
length_plus_group Is the last bin a plus group
tolerance Tolerance for rescaling proportions
years Years for which there are observations
selectivities The labels of the selectivities
process_errors Process error
time_step_proportion Proportion through the mortality block of the time step when the observation is evaluated

@observation[label].type=process_proportions_by_category

min_age Minimum age

max_age Maximum age
time_step The label of time-step that the observation occurs in
age_plus Use age plus group
years Years for which there are observations
selectivities The labels of the selectivities
categories2 Target Categories
selectivities2 Target Selectivities
process_errors Process error
process The label of the process for the observation
process_proportion Proportion through the mortality block of the time step when the observation is evaluated

@observation[label].type=proportions_by_category

min_age Minimum age
max_age Maximum age
time_step The label of time-step that the observation occurs in
age_plus Use age plus group
years Years for which there are observations
selectivities The labels of the selectivities
categories2 Target Categories
selectivities2 Target Selectivities
process_errors Process error
time_step_proportion Proportion through the time step to analyse the partition from

@observation[label].type=proportions_migrating

min_age Minimum age
max_age Maximum age
time_step The label of time-step that the observation occurs in
age_plus Use age plus group
years Years for which there are observations
process_errors Process error
ageing_error Label of ageing error to use
process Process label

@observation[label].type=tag_recapture_by_age

min_age Minimum age
max_age Maximum age
age_plus Use age plus group
years Years for which there are observations
categories2 The available categories in the partition
selectivities The labels of the selectivities
time_step The label of time-step that the observation occurs in
selectivities2 The categories of tagged individuals for the observation
process_errors Process error
detection Probability of detecting a recaptured individual
time_step_proportion Proportion through the mortality block of the time step when the observation is evaluated

@observation[label].type=tag_recapture_by_length

length_bins Length Bins

`plus_group`	Indicates if the last length bin is a plus group
`years`	Years for which there are observations
`time_step`	The label of time-step that the observation occurs in
`categories2`	The categories of tagged individuals for the observation
`selectivities`	The labels of the selectivities used for untagged categories
`selectivities2`	The labels of the tag category selectivities
`process_errors`	Process error
`detection`	Probability of detecting a recaptured individual
`dispersion`	Over-dispersion parameter (phi)
`time_step_proportion`	Proportion through the mortality block of the time step when the observation is evaluated

@penalty *label* Define an object of type *penalty*

`label`	The label of the penalty
`type`	The type of penalty

@penalty[label].type=process

`multiplier`	The penalty multiplier
`log_scale`	Indicates if the sums of squares is calculated on the log scale

@process *label* Define an object of type *process*

`label`	The label of the process
`type`	The type of process
`print_report`	Indicates if a process report should be generated for this process

@process[label].type=ageing

`categories`	The labels of the categories

@process[label].type=maturation

`from`	List of categories to mature from
`to`	List of categories to mature too
`selectivities`	List of selectivities to use for maturation
`years`	The years to be associated with rates
`rates`	The rates to mature for each year

@process[label].type=mortality_constant_rate

`categories`	List of categories labels
`m`	Mortality rates
`time_step_ratio`	Time step ratios for the mortality rates
`selectivities`	List of selectivities for the categories

@process[label].type=mortality_event

`categories`	Categories
`years`	Years in which to apply the mortality process
`catches`	The number of removals (catches) to apply for each year
`u_max`	Maximum exploitation rate (*Umax*
`selectivities`	List of selectivities
`penalty`	The label of the penalty to apply if the total number of removals cannot be taken

@process[label].type=mortality_event_biomass

`categories`	Category labels

`selectivities` The labels of the selectivities for each of the categories
`years` Years in which to apply the mortality process
`catches` The biomass of removals (catches) to apply for each year
`u_max` Maximum exploitation rate ($Umax$
`penalty` The label of the penalty to apply if the total biomass of removals cannot be taken

`@process[label].type=mortality_holling_rate`

`prey_categories` Prey Categories labels
`predator_categories` Predator Categories labels
`is_abundance` Is vulnerable amount of prey and predator an abundance [true] or biomass [false]
`a` parameter a
`b` parameter b
`x` This parameter controls the type of functional form, Holling function type 2 (x=2) or 3 (x=3), or generalised (Michaelis Menten, x,=1
`u_max` Maximum exploitation rate ($Umax$
`prey_selectivities` Selectivities for prey categories
`predator_selectivities` Selectivities for predator categories
`penalty` Label of penalty to be applied
`years` Years in which to apply the mortality process

`@process[label].type=mortality_initialisation_event`

`categories` Categories
`catch` The number of removals (catches) to apply for each year
`u_max` Maximum exploitation rate ($Umax$
`selectivities` List of selectivities
`penalty` The label of the penalty to apply if the total number of removals cannot be taken

`@process[label].type=mortality_initialisation_event_biomass`

`categories` Categories
`catch` The number of removals (catches) to apply for each year
`u_max` Maximum exploitation rate ($Umax$
`selectivities` List of selectivities
`penalty` The label of the penalty to apply if the total number of removals cannot be taken

`@process[label].type=mortality_instantaneous`

`categories` Categories for instantaneous mortality
`m` Natural mortality rates for each category
`time_step_ratio` Time step ratios for natural mortality
`selectivities` The selectivities to apply on the categories for natural mortality

`@process[label].type=mortality_prey_suitability`

`prey_categories` Prey Categories labels
`predator_categories` Predator Categories labels
`consumption_rate` Predator consumption rate
`electivities` Prey Electivities
`u_max` Umax
`prey_selectivities` Selectivities for prey categories
`predator_selectivities` Selectivities for predator categories
`penalty` Label of penalty to be applied
`years` Year that process occurs

@process[label].type=nop

@process[label].type=recruitment_beverton_holt

`categories` Category labels
`r0` R0
`b0` B0
`proportions` Proportions
`age` Age to recruit at
`ssb_offset` Spawning biomass year offset
`steepness` Steepness
`ssb` SSB Label (derived quantity
`b0_intialisation_phase` Initialisation phase Label that b0 is from
`ycs_values` YCS Values
`ycs_years` Recruitment years. A vector of years that relates to the year of the spawning event that created this cohort
`standardise_ycs_years` Years that are included for year class standardisation

@process[label].type=recruitment_constant

`categories` Categories
`proportions` Proportions
`age` Age
`r0` R0

@process[label].type=survival_constant_rate

`categories` List of categories
`s` Survival rates
`time_step_ratio` Time step ratios for S
`selectivities` Selectivity label

@process[label].type=tag_by_age

`from` Categories to transition from
`to` Categories to transition to
`min_age` Minimum age to transition
`max_age` Maximum age to transition
`penalty` Penalty label
`u_max` U Max
`years` Years to execute the transition in
`initial_mortality`
`initial_mortality_selectivity`
`loss_rate`
`loss_rate_selectivities`
`selectivities`
`n`

@process[label].type=tag_by_length

`from` Categories to transition from

`to`	Categories to transition to
`plus_group`	Use plus group for last length bin
`maximum_length`	The upper length when there is no plus group
`penalty`	Penalty label
`u_max`	U Max
`years`	Years to execute the transition in
`initial_mortality`	
`initial_mortality_selectivity`	
`selectivities`	
`n`	

`@process[label].type=tag_loss`

`categories`	List of categories
`tag_loss_rate`	Tag Loss rates
`time_step_ratio`	Time step ratios for Tag Loss
`tag_loss_type`	Type of tag loss
`selectivities`	Selectivities
`year`	The year the first tagging release process was executed

`@process[label].type=transition_category`

`from`	From
`to`	To
`proportions`	Proportions
`selectivities`	Selectivity names

`@process[label].type=transition_category_by_age`

`from`	Categories to transition from
`to`	Categories to transition to
`min_age`	Minimum age to transition
`max_age`	Maximum age to transition
`penalty`	Penalty label
`u_max`	U Max
`years`	Years to execute the transition in

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

`label`	Label
`steps`	The number of steps to take between the lower and upper bound
`lower_bound`	The lower bounds
`upper_bound`	The upper bounds
`parameter`	The system parameter to profile
`same`	A Parameter that are constrained to have the same value as the parameter being profiled

`@project` *`label`* Define an object of type *project*

`label`	Label
`type`	Type
`years`	Years to recalculate the values
`parameter`	Parameter to project
`multiplier`	Multiplier that is applied to the projected value

`@project[label].type=constant`

`values`	Values to assign to addressable

`@project[label].type=empirical_sampling`

`start_year`	Start year of sampling
`final_year`	Last year of sampling

`@project[label].type=log_normal`

`sigma`	Standard deviation of gaussian process on the log scale
`mean`	Mean of gaussian process

`@project[label].type=log_normal_empirical`

`mean`	Mean of gaussian process
`start_year`	Start year of sampling
`final_year`	Last year of sampling

`@project[label].type=user_defined`

`equation`	Equation to do a test run of

`@report` *`label`* Define an object of type *report*

`label`	The label for the report
`type`	The type of report
`file_name`	The File Name if you want this report to be in a seperate file
`write_mode`	The write mode

`@report[label].type=addressable`

`parameter`	Parameter to print
`years`	Years to print the addressable for
`time_step`	Time Step label

`@report[label].type=ageing_error_matrix`

`ageing_error`	Ageing Error label

`@report[label].type=catchabilities`

`@report[label].type=category_info`

`@report[label].type=category_list`

`@report[label].type=correlation_matrix`

`@report[label].type=covariance_matrix`

`@report[label].type=derived_quantity`

`@report[label].type=equation_test`

`equation`	Equation to do a test run of

`@report[label].type=estimate_summary`

`@report[label].type=estimate_value`

`@report[label].type=hessian_matrix`

`@report[label].type=initialisation_partition`

`@report[label].type=mcmc_covariance`

`@report[label].type=mcmc_objective`

`@report[label].type=mcmc_sample`

@report[label].type=m_p_d

@report[label].type=objective_function

@report[label].type=observation

`observation`	Observation label
`pearsons_residuals`	Print Pearsons Residuals
`normalised_residuals`	Print Normalised Residuals

@report[label].type=output_parameters

@report[label].type=partition

`time_step`	Time Step label
`years`	Years

@report[label].type=partition_biomass

`time_step`	Time Step label
`years`	Years
`units`	Units (Default Kgs

@report[label].type=partition_mean_weight

`time_step`	Time Step label
`years`	Years

@report[label].type=process

`process`	Process label that is reported

@report[label].type=project

`project`	Project label that is reported

@report[label].type=random_number_seed

@report[label].type=selectivity

`selectivity`	Selectivity name

@report[label].type=simulated_observation

`observation`	Observation label

@report[label].type=standard_header

@report[label].type=time_varying

@selectivity *`label`* Define an object of type *selectivity*

`label`	The label for this selectivity
`type`	The type of selectivity
`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=all_values

`v`	V

@selectivity[label].type=all_values_bounded

l	L
h	H
v	V

@selectivity[label].type=constant

c	C

@selectivity[label].type=double_exponential

x0	X0
x1	X1
x2	X2
y0	Y0
y1	Y1
y2	Y2
alpha	Alpha

@selectivity[label].type=double_normal

mu	Mu
sigma_l	Sigma L
sigma_r	Sigma R
alpha	Alpha

@selectivity[label].type=increasing

l	Low
h	High
v	V
alpha	Alpha

@selectivity[label].type=inverse_logistic

a50	A50
ato95	aTo95
alpha	Alpha

@selectivity[label].type=knife_edge

e	Edge
alpha	Alpha

@selectivity[label].type=logistic

a50	A50
ato95	Ato95
alpha	Alpha

@selectivity[label].type=logistic_producing

l	Low
h	High
a50	A50
ato95	Ato95
alpha	Alpha

@length_weight *label* Define an object of type *length_weight*

label	The label of the length-weight relationship
type	The type of the length-weight relationship

@length__weight[label].type=basic

a	The *a* parameter in the basic length-weight relationship
b	The *b* parameter in the basic length-weight relationship
units	Units of measure (tonnes, kgs, grams

@length__weight[label].type=none

@time_step *label* Define an object of type *time_step*

label	The label of the timestep
processes	The labels of the processes for this time step in the order that they occur