CASAL2 User Manual

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

NIWA Technical Report 139
ISSN 1174-2631
2019

CASAL2 User Manual (modified 2019-12-12) for use with
`casal2-v2019-12-12 (rev.  8ca79ad)`

Citation: S. Rasmussen, C. Marsh, I. Doonan, A. Dunn, K. Large (2019). CASAL2 User Manual, v2019-12-12 (rev. 8ca79ad). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 139*. 215 p.

Contents

1. Introduction

1.1. About CASAL2

CASAL2 is NIWA's open-source integrated assessment tool for modelling population dynamics of marine species, including fishery stock assessments. CASAL2 expands functionality and maintainability over its predecessor, CASAL. CASAL2 can be used for quantitative assessments of marine populations, including fish, invertebrates, marine mammals and seabirds.

CASAL2 software implements a generalised age-structured marine population model that allows a great deal of choice in specifying the population dynamics, parameter estimation, and model outputs. CASAL2 is designed for flexibility. It can implement an age-structured model for a single population or multiple populations using user-defined categories such as area, sex and maturity. However, these structural elements are generic and not predefined, but are easily constructed. CASAL2 models can be used for a single population with a single anthropogenic event (in a fish population model this would be a single fishery), or for multiple species and populations, areas, and/or anthropogenic or exploitation methods, and including predator-prey interactions.

The time period and annual cycle of CASAL2 is defined by the user. Observational data used can be from many different sources, for example removals-at-size or -age from an anthropogenic or exploitation event (e.g., fishery or other human impact), research survey and other biomass indices, and mark-recapture data. Model parameters can be estimated using penalised maximum likelihood or Bayesian methods.

As well as generating point estimates of the parameters of interest, CASAL2 can calculate likelihood or posterior profiles and can generate Bayesian posterior distributions using Monte Carlo Markov Chain methods. CASAL2 can project population status into the future using deterministic or stochastic population dynamics, or simulate observations from a set of given model structures

1.2. Citing CASAL2

A suitable reference for this document is S. Rasmussen, C. Marsh, I. Doonan, A. Dunn, K. Large (2019). CASAL2 User Manual, v2019-12-12 (rev. 8ca79ad). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 139*. 215 p.

And the peer-reviewed journal article reference for CASAL2 is (Doonan et al., 2016).

1.3. Software license

This program and the accompanying materials are made available under the terms of the GNU General Public License version 2 which accompanies this software (see Section 17).

1.4. 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`.

1.5. System requirements

CASAL2 is available for most IBM compatible machines running 64-bit Linux and Microsoft Windows operating systems.

Several of CASAL2's tasks are highly computer intensive and a fast processor is recommended. Depending on the model implemented, some of the CASAL2 tasks can take a considerable amount of processing time (minutes to hours), and in extreme cases may take several days to complete 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 the 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.

1.6. 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.

However, 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 15).

1.7. Getting help

CASAL2 is distributed as unsupported software. The Development Team would appreciate being notified of any problems or errors in CASAL2, please contact the `Casal2 Development Team`. See Section 16.2 for the recommended template for reporting issues.

1.8. 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` 8.1.0 (`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 as an *addressable parameter* to the model — to be reported, or to be an input into another process (e.g. 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 the time-step in which they occur. Hence, time-steps can be used to break processes into groups, and assist in defining the timing of the observations within the annual cycle. The 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, comprising 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 MCMC 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 CASAL2's 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 and tag recapture observations. Estimation generates values for each of the estimated parameters that are the best fit to the data, i.e., where each expected value is ‘close’ (in some mathematical sense) to the corresponding observed 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 communicate helpful and informative messages generated from the source code to the screen as the application runs, 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 a console window (i.e., the command line) on Microsoft Windows or from a terminal window on Linux. CASAL2 uses information from input data files -- the *input configuration file* being the main input file.

The input configuration file is required and defines the model structure and processes, the observations and 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 be included depending on the run mode and what information is required. For example, an input file can define the initial parameter values for estimation, or values from which to simulate observations, or values from which to run projections.

Command line arguments are used to specify the actions or *tasks* of CASAL2, e.g., to run a model with a set of parameter values, to estimate parameter values (either point estimates or MCMC), to project quantities, or to simulate observations. Hence, the *command line arguments* define the *task*. For example, `-r` is the *run* mode, `-e` is the *estimation* mode, and `-m` is the *MCMC* mode. The *command line arguments* are described in Section 3.4.

3.1. Using CASAL2

To use CASAL2, open a console window (i.e. the command prompt) window on Microsoft Windows or a terminal window on Linux. Navigate to the directory where the model input configuration files are located. Then enter `casal2` with arguments for a specific mode to start the CASAL2 mode running; see Section 3.4 for the list of possible arguments. CASAL2 will print output to the screen.

The CASAL2 executable and shared libraries (files with extension `.dll` or `.so`) must either be in the same directory as the input configuration files or in one of the directories in your operating system's `PATH` environment variable. The CASAL2 installer will update `PATH`; see your operating system documentation for help displaying or modifying `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 estimated variables (the estimation section),
- the observations and their associated properties and likelihoods (the observation section), and
- the output values and reports that CASAL2 will output (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, particularly if you have a model that has many partitions or timesteps, and can result in a input configuration file that is long and difficult to navigate.

To aid in making the model configuration more readable and flexible, the input configuration file command `!include "filename"` can be used (Figure 3.1). This command specifies that another file, `filename`, be read and processed, exactly as if its contents had been inserted into the main input configuration file at that point. The file name must be the complete file name with extension, and can use either the relative or absolute path as part of its name. Included files can also contain `!include` commands. See Section 12 for more detail.

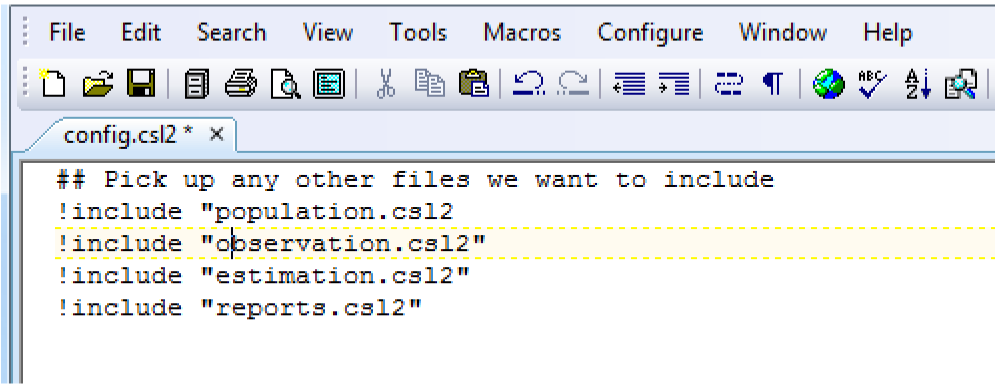

Figure 3.1: Example of using the input configuration file command `!include "filename"`.

3.3. Redirecting standard output

CASAL2 uses the `standard output` stream to display runtime information. The standard error stream is used by CASAL2 to output the program exit status and runtime 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] > run.out) >& run.err &
```

It may be useful to redirect the standard input, especially if you're using CASAL2 inside a batch job, i.e.

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

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

```
casal2 [arguments] > run.out
```

And, on some Microsoft Windows systems (e.g., Windows 10), you can redirect to both standard output and standard error, using the syntax

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

CASAL2 outputs header information to the output (Figure 3.2). 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). This information can be used to track outputs as well as identifying the version of CASAL2 used to run the model.

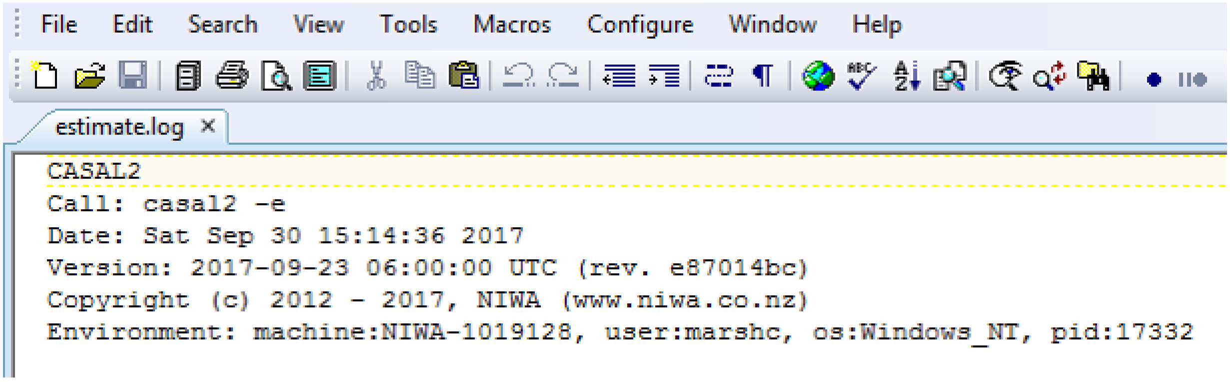

Figure 3.2: Example of output file header information.

3.4. Command line arguments

CASAL2 is called using:

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

where

-c *config_file* Define the input configuration file for CASAL2 (if this argument is omitted, the default input configuration file is `config.csl2`)

and where *task* must be one of the following ([] indicates a secondary label to call the task, e.g. **-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 starting parameter values from the file specified with argument `-i` *`filename`*

-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 starting parameter values from the file specified with the argument `-i` *`filename`*

-p [--profiling] Do a likelihood *profile* using the parameter values in the input configuration file as the starting point, or optionally with the starting parameter values from the file specified with the argument `-i` *`filename`*

-m [--mcmc] Do an *MCMC* chain using the values in the input configuration file as the starting point for the parameters to be estimated, or optionally with the starting parameter values from the file specified with the argument `-i` *`filename`*

-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 starting parameter values from the file specified with the argument `-i` *`filename`*

-s [--simulation] *number* *Simulate* the *number* of observation sets using values in the input configuration file as the parameter values, or optionally with the parameter values from the file specified with the argument `-i` *`filename`* and where the following optional arguments [*options*] may be specified

-i [--input] *filename* *Input* one or more sets of free (estimated) parameter values from `filename` (see Section 11 for details about the format of `filename`)

-o [--output] *filename* *Output* a report of the free (estimated) parameter values in a format suitable for `-i filename` (see Section 11 for details about the format of `filename`)

-g [--seed] *seed* Initialise 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 to print `@report` in tabular format (see Section 7)

--single-step Run with `-r` to 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] *object type* *Query* an object type to print an extract of the object description and parameter definitions. An object can be defined as `block.type`, e.g. `casal2 --query process.recruitment_constant` will query the constant recruitment block.

3.5. Constructing the CASAL2 input configuration files

The model definition, characteristics, parameters, observations, and reports are specified in input configuration files:

- Population input (Section 4) specifies the model structure, population dynamics, and other associated parameters;
- Estimation input (Section 5) defines the objective function, parameters of the model, and the method of estimation (point estimates, Bayesian posteriors, profiles, etc.);
- Observation input (Section 6) specifies the observations data used in the model and describes how the observed values should be formatted, how CASAL2 calculates the expected values, and the likelihoods applied for each set of observations; and
- Report input (Section 7) specifies the output.

The command and subcommand syntax to be used in each of these configuration sections are listed in Sections 8 (Population), 9 (Estimation), 10 (Observation) and 11 (Report).

3.5.1. Commands

CASAL2 has a range of commands that define the model structure, processes, parameters, observations, and how tasks are carried out. There are three types of commands

- Commands that have an argument and do not have subcommands (for example, `!include filename`)
- Commands that have a label and subcommands (for example `@process` must have a label and has subcommands)
- 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 whitespace (tabs or spaces) 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

CASAL2 subcommands define options and parameter values related to a particular command. Subcommands always take an argument which is one of a specific *type*. The argument *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 the expected results.

The argument type for a subcommand can be:

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., a double)
constant vector	a vector of real numbers (i.e., a vector of doubles)
estimable	a real number that can be estimated (i.e., a double)
estimable vector	a vector of real numbers that can be estimated (i.e., a vector of doubles)
addressable	a real number that can be referenced but not estimated (i.e., an addressable double)
addressable vector	a vector of real numbers that can be referenced but not estimated (i.e., a vector of addressable doubles)
string	a categorical (string) value
string vector	a vector of categorical values.

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

Integers must be entered as whole numbers without decimal points (i.e., if `year` is an integer then it is specified as `2008`, not `2008.0`)

Arguments of type integer vector, constant vector, estimable vector, addressable vector, or categorical vector must contain one or more entries on a row, separated by whitespace (tabs or spaces). Arguments of type integer range must contain a colon (`:`) and no whitespace (tabs or spaces).

Parameters are defined in the population section and can be specified as estimable with the subcommand type `estimable` or `estimable vector`. These parameters will be estimated if specified as such in the estimation section. If an estimable parameter is not specified in the estimation section it will instead be treated as a constant (or constant vector). In other words, only estimable parameters can be estimated and the parameter command must explicitly specify that the parameter is estimable with the `estimable` or `estimable vector` subcommand type.

Parameters defined as addressable with the subcommand type `addressable` or `addressable vector` are usually derived quantities and are not directly estimable. As such, they can be referenced by various processes, or have priors and/or penalties associated with them, but they do not directly contribute to any estimation within the model.

3.5.3. The command block format

Each command block consists of a single command which starts 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`	`@command argument`	`@command` *`label`*
`subcommand argument`	`subcommand argument`	`subcommand argument`
`subcommand argument`	`subcommand argument`	`subcommand argument`
.	.	.
.	.	.
etc.	etc.	etc.

Blank lines are ignored, as is extra whitespace (tabs and spaces) between arguments. However, to start command block the `@` character must be the first character on the line and must not be preceded by any whitespace. Each input file must end with a carriage return.

There is no indicator of the end of a command block. Each command block is delimited by the end of the file, the end of the section, or the start of the next command block (which is marked by the `@` on the first character of a line). The `!include` command is the only exception to this rule (see Section 12 for details of the use of `!include`).

Commands, subcommands, and arguments in the input configuration files are not case sensitive. However, labels and variable values are case sensitive. On Linux, filenames and paths are case sensitive (i.e., when using `!include` *`filename`*, the argument *`filename`* will be case sensitive).

3.5.4. Commenting out lines

Text on a line that starts with the symbol # is considered to be a comment and is ignored. To comment out a group of commands or subcommands, use # at the beginning of each line to be ignored.

Alternatively, to comment out an entire block or section, use /* at the beginning of a line to start the comment block, then end the block with */. All lines (including line breaks) between /* and */ inclusive are ignored.

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

3.5.5. Determining CASAL2 parameter names

When CASAL2 processes the input configuration file it translates each command block and each subcommand block into a CASAL2 object, each with a unique parameter 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 that parameter when, e.g., estimating, applying a penalty, projecting, time varying, or profiling. For example, the parameter name of the Natural Mortality rates subcommand `m` of the command `@process` with the label `NaturalMortality` is category related and so, the syntax to reference all `m` related categories is

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

The syntax to specify a single category to which the natural mortality process is applied is

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

All labels (parameter names) are user specified. As such, naming conventions are non-restrictive and can be model specific.

3.6. Single-stepping CASAL2

Single-stepping means CASAL2 can 'pause' after each year in the annual cycle during a model run, write reports, then wait and process user input of updated estimable parameters for the next year (see the command line argument `--single-step`).

This enables CASAL2 to implement models for management simulations or scenarios that require feedback and can be used, for example, in operational management procedures (OMPs). The single-stepping process can be automated using **R**, so that CASAL2 may be used with **R** to update input harvest values (e.g., catches from a fishery in a fisheries model) to evaluate a particular harvest control rule.

3.7. CASAL2 exit status values

When CASAL2 is run, it will complete its task successfully or output errors. CASAL2 will return a single exit status value 'completed' to the standard output. Error messages will be printed to the console. When input file configuration errors are found, CASAL2 will print error messages, along with the associated filename(s) and line number(s) 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 of the population dynamics. It describes the model structure, the population partitions and categories, the population processes (e.g., recruitment, ageing, migration, and mortality), the selectivities, and the associated parameters.

The population section includes:

- The population structure, the categories and ages in an age-based model;
- The initialisation process, the state of the partition at the start of the first year;
- The years over which the model runs, the start and end years of the model;
- The annual cycle, the number of time steps and the processes that are applied in each time step;
- The specification of and the parameters for the population processes, processes that add or remove individuals from a partition, or shift individuals between ages and categories in a partition;
- The selectivities;
- The parameters, their definitions, initial values, prior distributions, and other characteristics; and
- Derived quantities, e.g., mature biomass to include in density-dependent processes such as the spawner-recruit relationship

4.2. Population structure

The basic structure of the 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 assumes an annual cycle.

Each year is defined by one or more time steps, with at least one process occurring in each time step. Each time step can represent a specific period of the calendar year, or it can be an abstract sequence of events.

The division of the year into time steps allows the user to specify the exact order in which processes and observations occur throughout the year. The user specifies the time step in which each process occurs. If more than one process occurs in the same time step, the order in which to apply each process is specified in the `@time_step` block.

The mortality processes are grouped into a mortality block: in every time step, a mortality block (a group of consecutive mortality-based processes) exists in which individuals are removed from the partition (see Section 4.4.2).

The state is the current status of the population at any given time. The state can change one or more times in each time step in each year. The state object must contain sufficient information to determine how the population changes over time, given a model and a complete set of parameters.

The state can undergo a number of possible changes during the annual cycle, called transitions. Transitions are applied by processes. Transitions include recruitment, natural mortality, fishing mortality, ageing, migration, tagging events, and maturation. These transitions are repeated for each year of the model, although some processes can be specified to occur in a subset of years only.

The key element of the state is the partition. The partition separates the total number of individuals in the population into different ages, lengths, and/or categories. The categories include sex, maturity state, area, and species. CASAL2 has no predefined categories; *all* categories are defined by the user, which differs from CASAL (Bull et al., 2012).

The partitions can be conceptualised as a matrix, where each row represents a category and the columns are the age classes (Figure 4.1). Each row represents all individuals that category.

The names of categories are user defined. There must be at least one category defined for each model. The model ages are a sequence from age_{min} to age_{max}, with the last age optionally a plus group. The age-length relationship for each category must also be defined for an age-based model, although this relationship could be defined as "none". An example of four categories based on sex and area is:

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

Consider a model of a fish population with a mature fishery and a non-spawning fishery. Assume that the non-spawning fishery occurs in the non-spawning area. The mature fish then migrate to the spawning area, where the spawning fishery occurs. At the end of spawning, these fish, along with the recruits from the previous year, migrate back to the non-spawning area. The fish population can be represented with partitions by age, sex, maturity, and area (spawning and non-spawning areas). So the partition has 8 rows of numbers-at-age, for 2 sexes $\times$ (mature or immature) $\times$ 2 areas.

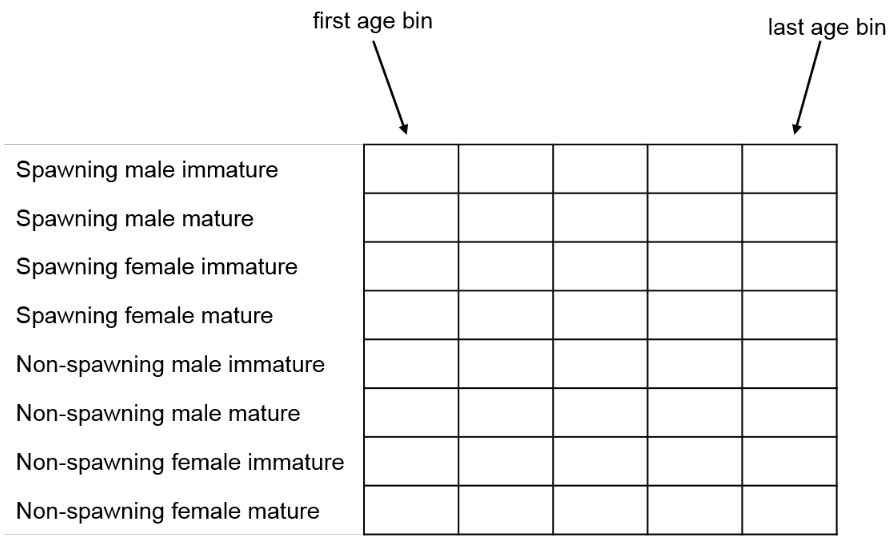

Figure 4.1: A visual representation of a partition.

For this example four time steps are defined and labelled 1 through 4: `step1` for the non-spawning fishery period, `step2` for the migration to the spawning area, `step3` for the spawning fishery period, and `step4` for recruitment and migration back to the non-spawning area. The default order of processes within a time step has migrations occurring before fisheries (TODO: check this), so that the processes in steps 2 and 3 could have occurred in one time step. Other details that describe the population structure are also linked to time steps, such as proportion of natural mortality occurring in each time step and in which time step the observations occur.

The definition and ordering of processes in multiple time steps can be used to represent complex dynamics, with the intermingling of multiple species and stocks, migration patterns occurring over multiple areas, and/or multiple sources of anthropogenic impact using a range of methods which cover different areas and times. However, the complexity of a stock structure definition is constrained by the available data. It is challenging to use a complex structure to model a population when there are no observations to support that structure. For information on how to define categories and use the shorthand syntax see Section 14.2.

The model is run from the start year through the final year. It can also be run past the final year to project the state of the population through the final projection year.

To specify a model with two categories, male and female, population ages 1-20, with the last age a plus group, three time steps, and sex-specific age-length relationships, the `@model` and `@categories` blocks are:

```
@model
start_year 1901
final_year 2000
projection_final_year 2010
base_weight_units tonnes
min_age 1
max_age 20
age_plus_group true
initialisation_phases Equilibrium_phase
time_steps step1 step2 step3

@categories
format sex
names male female
age_lengths male_growth female_growth
```

4.3. The state object and the partition

The key component of the state object is the partition, a matrix that stores the numbers of individuals at age or length for each category. A category represents a group of individuals that have the same attributes, e.g., life histories characteristics, growth rates, etc.

- Sex (male or female)
- Area
- Maturity (immature or mature)
- Growth path
- Tagging event
- Stock
- Species

A stock is defined as a population of individuals which recruits to that population. Maturity can either be defined as a separate category in the partition, or calculated from the population at the time required; see Section 4.11 for the treatment of maturity when maturity is not a category in the partition.

Each CASAL2 model requires:

- The minimum and maximum population ages
- Whether the maximum age is a plus group
- The names of all of the categories

The age range is sequential by 1 starting with the minimum age through the maximum age.

CASAL2 allows categories of the partition to exist for a subset of years of a model. This feature enables more efficient computations when models contain categories that do not persist over all model years. A model may define one-off processes that transition individuals from one category into another in a subset of the model initialisation phases or years (e.g., tagging events). Excluding categories for certain years can be more efficient as CASAL2 will not initialise these categories or apply processes to categories in years or time steps in which they do not exist.

The structure of the partition is defined in a configuration block with the `@categories` block (Section 4.2).

Derived quantities are another important component of the state object. An example of a derived quantity is spawning stock biomass (SSB; the biomass of [female] spawning fish calculated at the mid point of the spawning season). CASAL2 calculates derived quantities using the command `@derived_quantity`, which may be required for some processes. In fisheries stock assessment models, a recruitment process which includes 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 annual cycle
- The mortality blocks
- The initialisation phases
- The model run years
- The projection years

4.4.1. The 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 specific time periods and processes. Any number of processes can occur within each time step, in any order, although there are restrictions for mortality-based processes (see Section 4.4.2); processes can occur multiple times within each time step. Time steps are not implemented during the initialisation phases (effectively there is only one initialisation time step), and the annual cycle in the initialisation phases can be different from the annual cycle specified for the model years.

4.4.2. The mortality blocks

There is an associated *mortality block* for every time step in the annual cycle. 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 4.6) are evaluated.

A mortality block is defined as a consecutive sequence of mortality processes within a time step. The mortality processes are described in Subsection 4.5.3.

CASAL2 requires that each time step has exactly one mortality block. Either all of 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 output an error if more than one mortality block occurs in a single time step. Use separate time steps to define a sequence of mortality blocks.

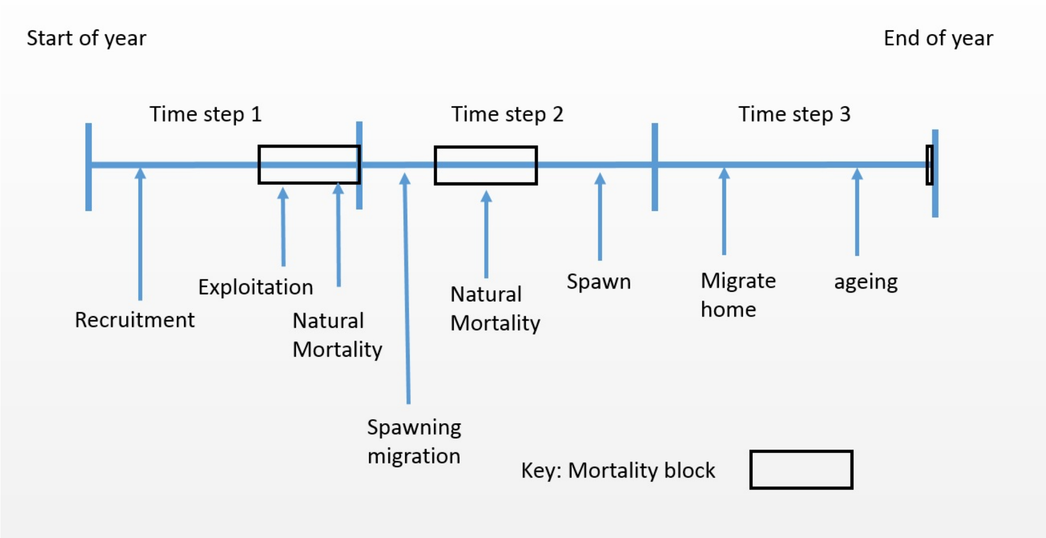

Figure 4.2: A example sequence for an annual cycle.

4.4.3. The initialisation phases

Initialisation is the process of determining the model starting state. The initial state can be equilibrium/steady state or some other initial state for the model (e.g., exploited), prior to the start year of the model.

There are multiple options for partition initialisation in CASAL2, including

- Iterative
- Derived
- Cinitial
- Fixed

Model initialisation can also occur in several phases, each of which can use a different method. The initialisations are performed in sequence. At the end of all of the initialisation phases, CASAL2 then runs through the model years applying the user-defined processes in each time step in the annual cycle.

The multi-phased initialisation allows for flexibility in the number and type of initialisations, for initialising a non-equilibrium starting state, or applying simple processes before applying 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. An initialisation phase can include other processes with the `insert_processes` subcommand.

In each initialisation phase, the processes defined for that phase are applied and used as the starting point for the following phase or, if it is the last phase, the start year of the model.

The *first* initialisation phase is always initialised with each age and category set to zero. Care must be taken 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 an initialisation has converged. A second initialisation phase can be added for 1 year, with the same processes applied as in the first phase. The state at the end of the first and second phase is then output. If these states are identical, then it is likely that the initialisation has converged to an equilibrium state.

The subcommands for including or excluding processes are `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
```

For example, this subcommand could be used in a `@time_step` labelled `Oct_Nov`, which includes the `@process` labelled `predationInit`, and before the `@process` labelled `Instantaneous_Mortality`,

```
insert_processes Oct_Nov(Instantaneous_Mortality)=predationInit
```

To include a process at the end of the time step:

```
insert_processes Oct_Nov()=predationInit
```

To exclude a process from an initialisation phase, use the subcommand `exclude_processes` in a command `@initialisation_phase`,

```
exclude_processes Instantaneous_Mortality
```

This command removes the process labelled `Instantaneous_Mortality` during that particular initialisation phase.

4.4.3.1. Iterative Initialisation

The `Iterative` initialisation is a general solution for initialising the model. The iterative method can be slow to converge, depending on the model, but can 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 increase the model output, and the number of iterations should be chosen to be large enough to allow the population state to fully converge. A period of about two generation times is recommended to ensure convergence. CASAL2 can be configured to report convergence statistics that can assist in determining convergence properties.

In addition, the iterative initialisation phase can optionally be stopped early if user-defined convergence criteria is met. For a list of supplied years in the initialisation phase, the convergence criteria is 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 the user-defined value of 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 initialisation define:

- The number of initialisation phases,
- The number of years in each phase, and
- The processes to apply in each phase, where the default processes are those applied in the annual cycle

An example with one initialisation phase:

```
@model
...
initialisation_phases Iterative_initialisation

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

4.4.3.2. Derived Initialisation

The `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/steps/units?, so it is computationally faster than the iterative initialisation phase. Under some process combinations (e.g., one-way migrations) this initialisation does not calculate the exact equilibrium partition. When using this initialisation, confirm that the partition has reached an equilibrium state by either comparing with an iterative initialisation, or by adding a second iterative initialisation phase with a limited number of iterations for comparison.

An example with one initialisation phase:

```
@model
...
initialisation_phases Equilibrium_initialisation

@initialisation_phase Equilibrium_initialisation
type derived
```

4.4.3.3. Cinitial Initialisation

The `Cinitial` initialisation is used only as a second or greater phase initialisation, and can only be applied after Derived or Iterative initialisation phases. This initialisation can be a method for estimating the non-equilibrium state of population if there is exploitation before the data start. The estimated Cinitial factors shift the initial population away from an equilibrium state prior to the start year. It would be helpful 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 with two initialisation phases:

```
@model
...
initialisation_phases Iterative Cinitial

@initialisation_phase Iterative
type iterative
years 10
lambda 0.0001
convergence_years 10 20

@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 with the 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
```

4.4.3.4. Fixed Initialisation

The `Fixed` initialisation uses a user-defined table as the initial partition numbers-at-age prior to the start year. Models can be initialised by specifying the numbers-at-age for each category. When initialising models with this type, undefined behaviour may be result if the model applies processes that require derived quantities to be calculated in the initialisation phase.

An example with one initialisation phase:

```
@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
```

4.4.4. Model run years

Following initialisation, the model then runs over the user-defined years, from `start_year` to `final_year`. For this part of the model, the annual cycle can be broken into multiple time steps per year, and observations can be associated with specific time steps.

Processes are applied in the order specified within each time step. These processes can be the same or different from the processes specified for the initialisation phases.

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` to the end of year `final`. The projection then extends the run time up to the end of year `project_final_year`.

The model properties must be specified:

- The number of time steps and the processes applied in each
- The first year, the model start year
- The last year, the model final year
- The last projection year, the model projection final year

An example of the syntax:

```
@model
start_year 1972
final_year 2016
projection_final_year 2021
## Define the ages in the partition
min_age 1
max_age 30
age_plus true
base_weight_units tonnes
initialisation_phases Equilibrium_state
## Define the annual cycle
time_steps Sep_Feb Mar_May Jun_Aug

## Define the "rows" in the partition
## This is a single sex and area population
@categories
format stock
names HAK4
age_lengths age_size

@initialisation_phase Equilibrium_state
type derived

## Define the processes in the annual cycle
## A list of labels in each time step that correspond to a process
```

```
@time_step Sep_Feb
processes Recruitment Instantaneous_Mortality

@time_step Mar_May
processes Instantaneous_Mortality

@time_step Jun_Aug
processes  Ageing Instantaneous_Mortality
```

4.4.5. Projection years

The Projection functionality runs the model forwards, using stochastic and/or deterministic values for some population parameters, such as recruitments and catches.

The CASAL2 command to run the model in projection mode is `casal2 -f 1`. The number that follows the `-f` parameter indicates the number of projections to generate for each set of parameters supplied. This functionality allows for the exploration of many scenarios with a single set of parameters. The number of projections should be greater than 1 only if applying a projection type that is stochastic.

The `--tabular` flag should be used when running projections after a Bayesian analysis. This option will output a tabular report (see Section 7.21) which can then be analysed in **R**.

Projection years are after the model run years, and are defined as the `final_year + 1` through the `final_projection_year`.

For a projection run in CASAL2 the model is initialised and run through the model years from `start_year` to `final_year`. During this run mode CASAL2 stores all parameter values so that projection classes can allow parameters before `final_year` to be projected. The model then is re-run from `start_year` to `projection_final_year`, where any parameter can either be fixed or drawn from a stochastic distribution or process.

An example of when a parameter is projected before the projection phase has started is for year class parameters. The last few year class parameters may be poorly estimated, which depends on the quality and coverage of the composition data that could inform these parameters or the use of a recruitment index. Thus, users may assume that these parameters are unknown and apply projection methods for the future values.

CASAL2 has no default projection properties for parameters that are specified by year, e.g., year class strength parameters. The projections for these parameters must be specified using the `@project` command block. CASAL2 will produce errors if run in projection mode without a `@project` block for the `ycs_values` parameter being specified.

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
- user-defined

The projection classes available, and examples of their syntax, are in Section 4.13.

The subcommands `years` and `parameter` are common to all projection methods. The argument `multiplier` is a constant which is multiplied with the projected value after it has been generated.

Note for the year class parameters: the definition of year applies to the `ycs_years`, not the model years. As defined in Section 4.5.1.2, `ycs_years` are offset between the time of spawning and when individuals are added to the partition.

4.5. Population processes

Population processes are processes that change the model state. These processes produce changes in the partition by adding and removing individuals, or by moving individuals between ages and/or categories.

The population processes include:

- recruitment,
- ageing,
- growth,
- maturation,
- mortality events (e.g., natural and fishing), and
- category transition processes, i.e., processes that move individuals between categories while preserving their age structure.

There are two types of processes: (1) processes that occur across multiple time steps in the annual cycle, e.g., `mortality_constant_rate` and `mortality_instantaneous`; and (2) processes that occur only within the time step in which they are defined. These processes are applied in the user-defined order when initialising the model, and then for a user-defined order in each year in the annual cycle.

4.5.1. Recruitment

Recruitment processes are defined as processes that add new individuals to the partition. CASAL2 has two options for recruitment processes, constant recruitment and the Beverton-Holt stock-recruitment relationship (Beverton and Holt, 1957).

In the recruitment processes, a number of individuals are added to a single age class within the partition, with the number determined by the type of recruitment process specified. If more than one category (of recruits?) is defined, then the proportion of recruits to be added to each category is specified by the `proportions` property, or multiple recruitment processes can be defined. For example, if recruiting to categories labelled `male` and `female`, then the proportions may be set to 0.5 and 0.5, so that half of the recruits are added to the male category and the other half to the female category.

Recruitment can differ between a spawning event or the creation of a cohort/year class. In a fisheries context, recruitment usually refers to individuals "recruiting" to a fishery. This definition is used because there is usually not a lot of information on younger age classes between the time of spawning and being vulnerable to a survey or fishery for data collection. Thus, the model configuration may specify the population for which data are available.

The offset between spawning and recruitment is parameterised either by the recruitment variable `age`, or `min_age`, which is the default value for the `age` property in the recruitment process. The

CASAL2 parameter `age` is the same as the CASAL parameter `y_enter`.

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

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

where $N_{y,a,j}$ is the numbers in year y and category j at age a, p_j is the proportion added to category j, and R_y is the total number of recruits in year y.

4.5.1.1. Constant recruitment

In the constant recruitment process the total number of recruits added in each year y in age a is R_y, with $R_y = R_0$ for all years

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

Constant recruitment is equivalent to a Beverton-Holt recruitment process with steepness (h) set to 1.

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

```
@process Recruitment
type constant_recruitment
categories male.immature female.immature
proportions 0.5 0.5
r0 500000
age 1
```

4.5.1.2. Beverton-Holt recruitment

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

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

where

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

and a is age, p_j is the proportion of recruits to enter category j, and `ssb_offset` is the number of years lag between spawning and recruitment.

Recruitment refers to recruitment into the population and may differ from the spawning event. See below on more information about `ssb_offset`. In general this parameter should not be specified by the user.

$SR(SSB_y)$ is the Beverton-Holt stock-recruit relationship parametrised by the steepness h, and based on Mace and Doonan (1988) parametrisation

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

The Beverton-Holt recruitment process requires a value for B_0 and SSB_y to calculate the number of recruits. A derived quantity (see Section 4.6) 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, which is defined by the parameter `b0_initialisation_phase`.

During initialisation the YCS multipliers are assumed to be equal to 1, and recruitment that happens in the initialisation phases that occur before and during the phase when B_0 is determined are assumed to have steepness $h = 1$ (i.e., in those initialisation phases, recruitment is equal to R_0).

Recruitment in the initialisation phases after the phase where B_0 was determined are calculated using the Beverton-Holt stock-recruit relationship. R_0 and B_0 have a direct relationship when there are no density-dependent processes in the annual cycle. Models can thus be initialised using B_0 or R_0.

The property `ssb_offset` should not be manually specified; CASAL2 determines `ssb_offset` by the order of ageing, recruitment, spawning, and the recruitment parameter `age`

- if the annual time step order is recruitment, ageing, spawning, then `ssb_offset` should equal `age` + 1, or
- if the annual time step order is spawning, ageing, recruitment, then `ssb_offset` should equal `age` - 1, or
- `ssb_offset` = `age`

There may be scenarios where the user will input these values, e.g., if there are multiple ageing processes in the annual cycle. CASAL2 does not have functionality to accommodate this situation, so in this case `ssb_offset` would be manually defined.

There are two variants of this process and they refer to how the stock recruitment residuals or YCS_{ycs_year} are parametrised. This parametrisation can either be in natural space as year class strength (YCS) multipliers, or in log space as recruitment deviations. Due to the difference in terminology, these variants are implemented in two separate processes, `recruitment_beverton_holt` and `recruitment_beverton_holt_with_deviations`, respectively.

YCS (YCS_y)

The YCS parameter (`ycs_years`) is defined in Equation (4.5). The parameter `ycs_values` is referenced by the `ycs_years` parameter and is important to note when defining `@estimate`, `@project`, and `@time_varying` blocks for the parameter `ycs_values`. An example is at the end of the section.

A common practice when estimating YCS is to standardise using the Haist parametrisation, which was described by V. Haist. CASAL2 will standardise YCS only if subccommand `standardise_ycs_years` is defined. The model parameter `ycs_values` is a vector $\mathbf{Y}$, covering the years from `start_year - ssb_offset` to `final_year - ssb_offset`, as defined by the parameter `ycs_years`. The resulting 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`.

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

where S is the set of years from `standardise_ycs_years`. One effect of this parametrisation is that R_0 is then defined as the mean estimated recruitment over the set of years S, because the mean YCS multiplier over these years will always be one.

Typically `standardise_ycs_years` is defined to span the years over which YCS is reasonably well estimated. For years that are not well estimated, Y_y can be set to 1 for some or all years $y \in S$ (which is equivalent to forcing $R_y = R_0$ x $SR(SSB_y)$) by setting the lower and upper bounds of these Y values to 1. An exception to this might occur for the most recent YCS values, which the user may estimate but not include in the definition of R_0 (because the estimates may be based on too few data). One or more years may be excluded from the range of years for the averaging process of the Haist parametrisation.

The advantage of the Haist parametrisation is that a large penalty is not necessary to force the mean of the YCS parameter to be 1, although a small penalty should still be used to stop the mean of $\mathbf{Y}$ from drifting. These adjustments may improve MCMC performance. Projected YCS values are not affected by this feature. A disadvantage with this parametrisation in a Bayesian analysis is that the prior applies to Y, not YCS.

An example of the specification of a Beverton-Holt recruitment process, where individuals are added to the category "immature" at $age = 1$, and the number added 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 year class, with B_0 the value of the derived quantity at the end of the initialisation phase labelled `phase1`; and 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
```

Recruitment deviations, ε_y

Recruitment deviations represent the stock-recruitment relationship residuals in log space, with the link between YCS_y and ε_y

$$YCS_y = exp(\varepsilon_y - b_y \sigma_R^2/2) \tag{4.7}$$

where $\varepsilon_y \sim N(0, \sigma_R^2)$, σ_R^2 is the variance of the stock-recruitment residuals, and b_y is a bias correction defined by Methot Jr and Taylor (2011)

$$b_y = \left\{ \begin{array}{ll} 0, & \text{for } y \leq y_1^b \\ b_{max}(1 - \frac{y - y_1^b}{y_2^b - y_1^b}), & \text{for } y_1^b < y < y_2^b \\ b_{max}, & \text{for } y_2^b \leq y \leq y_3^b \\ b_{max}(1 - \frac{y_3^b - y}{y_4^b - y_3^b}), & \text{for } y_3^b < y < y_4^b \\ 0, & \text{for } y_4^b \leq y \end{array} \right\} \tag{4.8}$$

The ε_y values are normally distributed in log space and thus lognormal when back-transformed to the resulting stock-recruitment relationship YCS_y. Recent work has found that this transformation does not technically lead to the *a priori* assumption that the resulting YCS_y are lognormal. See Appendix A for more discussion.

The ramp function described above for the bias correction has the additional subcommands controlling the ramp

- y_1^b = `last_year_with_no_bias`
- y_2^b = `first_year_with_bias`
- y_3^b = `last_year_with_bias`
- y_4^b = `first_recent_year_with_no_bias`
- b_{max} = `b_max`

```
@process Recruitment
type recruitment_beverton_holt_with_deviations
categories immature
proportions 1.0
r0 500000
last_year_with_no_bias 1940
first_year_with_bias 1950
last_year_with_bias 2016
first_recent_year_with_no_bias 2018
b_max 0.85
b0_initialisation_phase phase1
steepness 0.75
age 1
ssb SSB_derived_quantity
deviation_years   1994   1995   1996   1997   1998   1999   2000   2001   2002   2003   2004   2005   2
deviation_values  0 -0.2 0.4  0  0  0  0  0  0  0  0  0  0
```

To specify a Beverton-Holt recruitment for each stock, the information required is:

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

When an `@initialisation_phase` (Section 4.4.3) type = `derived` is specified and the recruitment is defined by `b0`, then all categories must be specified in the `@recruitment` block. Usually in a recruitment processes only the categories that receive recruits need to be defined. For example, a population has a spawning area that is different from the area where recruits enter the population. An area-specific model could then be specified which contains spawning categories and recruiting categories. The recruiting categories would be specified in the subcommand `categories`, as these would be the categories receiving recruits.

If `@initialisation_phase`, `type=derived` is used, then all categories that are a part of that recruitment process need to be specified as well

```
@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
....
```

The `proportions = 0.0` for "spawn.male" and "spawn.female" are needed due to the way the derived initialisation phase works. The derived initialisation finds a solution for when `r0` = 1.0 based on an infinite geometric series for the plus group, and scales the initial partition by `r0`. Thus, if all categories are not specified, then those that are missed would not be initialised to true values and this could lead to inaccurate model outputs. This set-up extends to multiple-stock fisheries model configurations as well, where all of the categories that make up the stock need to be listed.

4.5.2. Ageing

The ageing process "ages" individuals, i.e., this process moves all individuals in the named categories j from one age class a to age class $a+1$, or accumulates them if the last age class is a plus group.

The ageing process is defined as,

$$\text{element}(a+1, j) \leftarrow \text{element}(a, j) \tag{4.9}$$

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

$$\text{element}(a_{\max}, j) \leftarrow \text{element}(a_{\max}, j) + \text{element}(a_{\max-1}, j). \tag{4.10}$$

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

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

Note: the ageing process is *NOT* applied by CASAL2 by default. As with other processes, CASAL2 will not apply a process unless it is 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.*

4.5.3. Mortality

There are 8 types of mortality processes available in CASAL2:

- constant rate,
- event,
- biomass-event,
- instantaneous,
- instantaneous retained,
- Hollings,
- initialisation, and
- a density-dependent relationship based on prey suitability.

These processes remove individuals from the partition, either as a rate, as a total number (abundance), as a biomass of individuals or, as a combination of these. CASAL2 does not (yet) implement the Baranov catch equation. However, instantaneous mortality is considered an approximation to the Baranov catch equation.

To apply both natural and biomass-event mortality, the mortality type `mortality_instantaneous` can be specified. Note that all mortality processes occur within the mortality block of a time step. See Section 4.4.2 for more information and definitions on mortality blocks.

4.5.3.1. Constant mortality rate

To specify a constant annual mortality rate (e.g. $M = 0.2$) for categories "male" and "female"

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

The total number of individuals removed from a category

$$D_{j,t} = \sum_{a} N_{a,j,t}[1 - \exp(-S_{a,j}M_{a,j}p_t)] \tag{4.11}$$

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

The mortality rate process requires a selectivity. To apply the same mortality rate over all age classes in a category, use a selectivity defined as $S_{a,j} = 1.0$ for all ages a in category j

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

Age-specific mortality rates can also be applied. For example, the hypothesis that mortality is higher for younger and older individuals and lowest when individuals are at their optimal fitness could be defined by using a double exponential selectivity (see Section 4.12)

```
@selectivity age_specific_M
type double_exponential
x0 7.06524
x1 1
x2 17
y0 0.182154
y1 1.43768
y2 1.57169
alpha 1.0

@process NaturalMortalityByAge
type mortality_constant_rate
categories male female
selectivities age_specific_M age_specific_M
m 1.0 1.0
```

In this definition `m` is set to 1.0 and the rate is described through the selectivity. This concept can be constructed similarly for other mortality methods such as `instantaneous_mortality`.

4.5.3.2. Event and biomass-event mortality

The event mortality and biomass-event mortality processes are applied in a similar manner, except that they remove a specified abundance (number of individuals) or biomass, respectively. These mortality processes can be used to define mortality events where the numbers of removals are known, e.g., fishing, 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 possible, while not exceeding the maximum exploitation rate.

Event mortality processes require a penalty to avoid estimating parameter values that will not allow the defined number of individuals to be removed. 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, with a similar penalty for biomass. 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:

First, calculate the vulnerable abundance for each category j in $1\dots J$ for ages $a = 1\dots A$ that are subject to event mortality

$$V_{a,j} = S_{a,j} N_{a,j} \tag{4.12}$$

and define the total vulnerable abundance V_{total} as

$$V_{total} = \sum_j \sum_a V_{a,j} \tag{4.13}$$

The exploitation rate to apply is

$$U = \begin{cases} C/V_{total}, & \text{if } C/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.14}$$

The number removed $R_{a,j}$ from each age a in category j is,

$$R_{a,j} = UV_{a,j} \tag{4.15}$$

For example, to specify an **abundance-based** fishing mortality process 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, the syntax is

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

and specifed similarly for a **biomass-based** fishing mortality process

```
@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
```

4.5.3.3. Instantaneous mortality

The instantaneous mortality process combines both natural mortality and event biomass mortality into a single process. This allows the simultaneous application of both natural mortality and anthropogenic mortality to occur across multiple time steps. This process applies half the natural mortality in each time step, then the mortalities from all the concurrent removals instantaneously, then the remaining half of the natural mortality. In fisheries models this is the most commonly used mortality process. It allows for multiple removal events in the case of a fisheries model multiple fisheries/fleets. There are a few constraints that a removal method can only occur in one time step, but you can have multiple removals to cover events during the year.

When instantaneous mortality is applied the following equations are used.

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

$$U_f = \frac{C_f}{\sum_a \bar{w}_a S_{f,a} n_a e^{-0.5tM_a}}$$

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

$$U_{f,obs} = max_a(\sum_k S_{k,a}U_k)$$

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

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

There is a maximum mortality pressure limit of $U_{f,max}$ for each method of removal f. So, no more than proportion $U_{f,max}$ can be taken from any element of the partition affected by removal method f in that time step. Clearly, $0 \leq U_{max} \leq 1$. It is an error if two removal methods, 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 mortality 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

$$n'_a = n_a exp(-tM_a)\left[1 - \sum_f S_{f,a}U_f\right]$$

As an example, to apply natural mortality of 0.20 across three time steps on both male and female categories, and with two methods of removals (fisheries) `FishingWest FishingEast` with respective catches (kg) known for years 1975:1977 (the catches are given in the `catches` table and information on selectivities, penalties and maximum exploitation rates are given in the `method` table), the syntax is,

```
@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
```

and for referencing catch parameters for use in projecting, time_varying and estimating, the syntax is,

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

where "method_label" is lower case from the catch or method table and continuing the example,

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

If you want weight to be calculated by empirical weight at age matrices as described in Section 4.9 the method table has an additional column that can call weight at age objects. For example

```
@age_weight jan_weight_at_age
type data
table data
year 1  2  3  4
1980 3.4 5.6 7.23  8.123
end_table

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

4.5.3.4. Instantaneous mortality with retained catch and discards

The instantaneous mortality retained process builds on the instantaneous mortality process (4.5.3.3) which has simultaneous applications of fishing and natural mortality, but with all catch-at-sea being landed, i.e., no discarding. The process mortality_instantaneous_retained allows for retained catch, discards, and also a mortality to be applied to discards, i.e., some are allowed to survive. The method for taking catch from the partition and constraints used are the same as in mortality_instantaneous.

The process was created to solve problems in the pot fishery for blue cod which has a minimum legal size and so some catch must be discarded at sea and some of these discards are expected to survive (based on some experimental work). There are length data taken at sea so that the catch selectivity can be estimated, and length and age data from the landed catch (retained) so that the retention selectivity can also be estimated. In this version of the process, discard mortality is specified by the user by defining a selectivity to represent mortality by age or length (e.g., constant of reverse logistic) and it is intended that it would not be estimated since there is no observation class for it yet. If discard mortality is not provided, it is assumed that all discards die. Landed catch, and both retained and catch selectivites must be specified.

Extending the example shown in instantaneous mortality process (4.5.3.3) to use retained weight instead of catch, the input commands would be:

```
@process FishingRetainedCatch
type mortality_instantaneous_retained
m 0.20
#natural mortality
time_step_ratio 0.42 0.25 0.33 # ratio of natural mortality in each of the tree time steps
selectivities One
#for natural mortality by age
categories male female
units kgs
```

```
table catches # two fisheries, West and East
year FishingWest FishingEast
1975 80000 111000 # these are now landed catch
1976 152000 336000
1977 74000 1214000
end table

table method #all discards die
method      category selectivity retained_selectivity u_max time_step penalty
FishingWest stock    westFSel    westRetainedSel      0.7   step1     CatchPenalty
FishingEast stock    eastFSel    eastRetainedSel      0.7   step1     CatchPenalty
end_table
```

If discard mortality is less than 1.0 is required, then use:

```
table method # 50% discard mortality
method      category selectivity retained_selectivity discard_mortality u_max time_step penalty
FishingWest stock    westFSel    westRetainedSel      DisMort           0.7   step1     CatchPen
FishingEast stock    eastFSel    eastRetainedSel      DisMort           0.7   step1     CatchPen
end_table

@selectivity DisMort
Type constant
c 0.5 #50% mortality of discards returned to the sea
```

See instantaneous mortality process (4.5.3.3) for referencing catch parameters and calculating weight using empirical weight by age matrices.

The report gives total catch, actual landed catch, and discards, without and with discard mortality; use the following

```
@report Mortality
type process
process Instantaneous_Mortality_Retained
```

In the following, fisheries are indexed by f, and a indexes both age and category combinations.

The total catch is found using by using a selectivity, $S_{f,a}$, in the same way as in the instantaneous mortality process. Retention, $R_{f,a}$, is defined by specifying a selectivity and it can be a function of length or age. The retained catch is the product of these two values, $R_{f,a}\,S_{f,a}$. If sex is in the partition, then there are potentially two retention curves, one for each sex. In general, there is a retention curve for each category in the partition. It does not apply to surveys. Discard mortality is also specified as a selectivity, $D_{f,a}$. The fraction of dead fish from fishing activity is $S_{f,a} * (R_{f,a} + (1.0 - R_{f,a}) * D_{f,a})$. If $D_{f,a}$ is 1.0, then all selected fish are dead, but if it is 0.0, then only retained fish are dead.

When the `mortality_instantaneous_retained` process is applied, the following equations are used:

- Total catch (catch-on-board), C_f, is calculated by (retained catch) * VF / VR, where VF is vulnerable retained biomass, j indexes categories and t is the proportion of M in the time step, and VF is the full vulnerable biomass, $VF = \sum_{a,j} \overline{w}_a S_{a,j} n_{a,j} \exp(-0.5 t M_{a,j})$.

- An exploitation rate (actually a proportion) is calculated for each fishery, as the total catch (retained + discards) divided by the selected biomass (VF above) using selectivity $S_{f,a}$,

$$U_f = \frac{C_f}{\sum_a \bar{w}_a S_{f,a} n_a \exp(-0.5tM_a)}$$

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

$$U_{f,obs} = max_a(\sum_k S_{k,a} U_k)$$

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

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

  There is a maximum mortality pressure limit of $U_{f,max}$ for each method of removal f. So, no more than proportion $U_{f,max}$ can be taken from any element of the partition affected by removal method f in that time step. Clearly, $0 \leq U_{max} \leq 1$. It is an error if two removal methods, 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 mortality pressures are recalculated. In this case the catch actually taken from the population in the model will differ from the specified catch, C_f.

- Discard numbers-at-age (including their share of natural mortality) is $S_{a,j}(1 - R_{a,j}) n_{a,j} \exp(-0.5tM_{a,j})$, and those that die at the end of the time step (updating the partition) are $D_{a,j} S_{a,j}(1 - R_{a,j}) n_{a,j} \exp(-tM_{a,j})$, where $D_{f,a}$ is the fraction that die on return to the sea.

- The partition is updated by removing landed catch, natural mortality, and discard mortality

$$n'_a = n_a \exp(-tM_a) \left[1 - \sum_f S_{f,a} U_f (R_{f,a} + D_{f,a}(1 - R_{f,a}))\right]$$

4.5.3.5. Holling's mortality rate

The density-dependent Hollings 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 the total (selected) abundance (or biomass) and some "predator" abundance (or biomass), and 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,

$$P(k,l) = S_{predator}(l) N_{predator}(k,l) \tag{4.16}$$

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

$$P_{total} = \sum_K \sum_L P(k,l) \tag{4.17}$$

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

$$V(k,l) = S_{prey}(l) N_{prey}(k,l) \tag{4.18}$$

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

$$V_{total} = \sum_K \sum_L V(k,l) \tag{4.19}$$

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

$$R_{total} = P_{total} \frac{a V_{total}^{x-1}}{b + V_{total}^{x-1}} \tag{4.20}$$

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,

$$U = \begin{cases} R_{total}/V_{total}, & \text{if } R_{total}/V_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.21}$$

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

$$R(k,l) = UV(k,l) \tag{4.22}$$

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
```

4.5.3.6. Initialisation-event mortality

Initialisation event mortality is a specific process 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, and mortality will not be applied outside of the initialisation phase. This process should not be embedded in the annual cycle. We advise that this process be used in conjunction with the `insert_processes` command in the `@initialisation_phase` block. 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 the `initialisation_mortality_event` has been specified in the initialisation phase `Predation_state` but not in the annual cycle.

4.5.3.7. Prey-suitability mortality

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

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

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

$$P(k,l) = S_{predator}(l) N_{predator}(k,l) \tag{4.23}$$

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

$$P_{total} = \sum_K \sum_L P(k,l) \tag{4.24}$$

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

$$V(g,k,l) = S_{prey}(l) N_{prey}(k,l) \tag{4.25}$$

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

$$V(g)_{total} = \sum_K \sum_L V(g,k,l) \tag{4.26}$$

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

$$A(g)_{total} = \frac{V(g)_{total}}{\sum_G V(i)_{total}} \tag{4.27}$$

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

$$R(g)_{total} = P_{total} CR \frac{A(g)_{total} E(g)}{\sum_G A(i)_{total} E(i)} \tag{4.28}$$

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

$$U(g) = \begin{cases} R(g)_{total}/V(g)_{total}, & \text{if } R(g)_{total}/V(g)_{total} \leq U_{max} \\ U_{max}, & \text{otherwise} \end{cases} \tag{4.29}$$

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

$$R(g,k,l) = U(g)V(g,k,l) \tag{4.30}$$

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

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

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

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

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

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

4.5.4. Transition By Category

This process moves individuals between categories. The CASAL2 partition is user-defined, and this type of process is used to move individuals between categories, 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).

4.5.4.1. 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, i.e., for every source category there is one target category,

$$N_{a,j} = N_{a,i} \times P_i \times S_{a,i} \tag{4.31}$$

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.

Note, to merge categories just repeat the "to" category multiple times.

An example, to specify a simple spawning migration of mature males from a western area to an eastern (spawning) area, 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.

4.5.5. Tag Release events

Tagging processes can be age or length based processes, where-by numbers of individuals are moved from an untagged category to a tagged category that the user has defined from the `@categories` block. Tag release processes can also account for initial 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. Individuals are deducted from the non-tagged categories and shifted into tagged categories. Often, the ages of tagged individuals are not known and length based tagging is the most commonly used tagging process.

Length-based tag release processes are more complicated, as CASAL2 needs to calculate the age-length matrix and the exploitation by each length-bin to then move the correct numbers-at-age based on the known lengths of release. CASAL2 also allows for initial tag loss. The algorithm that CASAL2 follows,

For each length bin (l) of the input vector of numbers at length ($\tilde{N}_l$) do the following

$$N_{l,j} = \sum_{a=1} N_{a,l,j} * S_a$$

where $N_{a,l,j}$ is the numbers at age and length for category j in the partition, and S_a is the selectivity at age a.

calculate the total numbers at length (T_l) across all source categories at length l taking into account the selectivities

$$T_l = \sum_{j=1} N_{l,j}$$

Calculate the transition rate for this length bin (u_l)

$$u_l = \tilde{N}_l / T_l$$

Check that we don't exceed some threshold, usually called a u_{max} which is analagous to the u_{max} in a mortality processes.

$$u_l = \begin{cases} u_{max}, & \text{if } u_l > u_{max} \textbf{ flag a penalty} \\ u_l, & \text{otherwise} \end{cases}$$

Calculate the numbers at age in this category that we are going to move by multiplying across the age-length matrix and storing this info by age, for each age we want to accumulated these for all length bins. then move the necessary

$$N_{a,j} += N_{a,l} * u_l$$

The syntax for an example of tag release by length process in CASAL2 follows,

```
@process 2005Tags_shelf
type tag_by_length
years 2005
from male.untagged+female.untagged
to male.2005  female.2005
selectivities ShelfselMale ShelfselFemale
penalty tagging_penalty
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 categories, based on the combination of growth rates and selectivity into tagged male and tagged female categories based on selectivities and growth rates between.

4.5.6. Tag Loss

Tag Loss is the process which accounts for tags being lost from tagged individual over time due to, for example, tag failure or tags getting knocked off. This process is applied as an instantaneous migration rate that can happen over multiple time steps in the annual cycle. This method assumes that when tags are lost 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 syntax for this relationship is,

```
@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
```

4.6. 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 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 over the categories (after applying a selectivity). Similarly for biomass Biomass derived quantities. Derived quantities are also calculated during the initialisation phases. Therefore, 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, e.g. 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 4.4.2 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`, and 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*), 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
```

4.7. Age-length relationship

The age-length relationship defines the length at age (and the weight at length, see Section 4.8) 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. When defining length-at-age in CASAL2, you must also define a length-weight relationship (see Section 4.8 below).The model can incorporate changes in length-at-age during the year âĂŤi.e., growth between fish birthdays âĂŤ by incrementing age as specified by the `time_step_proportions` parameter.

4.7.0.1. `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.

4.7.0.2. `von_bertalanffy`

$$\bar{s}(age) = L_\infty\left(1-\exp\left(-k\left(age-t_0\right)\right)\right) \tag{4.32}$$

4.7.0.3. `schnute`

$$\bar{s}(age) = \begin{cases} \left[y_1^b + (y_2^b - y_1^b)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right]^{1/b}, & \text{if } a \neq 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{1-\exp\left(-a(age-\tau_1)\right)}{1-\exp\left(-a(\tau_2-\tau_1)\right)}\right], & \text{if } a \neq 0 \text{ and } b = 0 \\ \left[y_1^b + \left(y_2^b - y_1^b\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right]^{1/b}, & \text{if } a = 0 \text{ and } b \neq 0 \\ y_1 \exp\left[\ln\left(y_2/y_1\right)\dfrac{age-\tau_1}{\tau_2-\tau_1}\right], & \text{if } a = 0 \text{ and } b = 0 \end{cases} \tag{4.33}$$

Note, 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 $k = a$).

4.7.0.4. `data`

There is an option for users to input empirical length at age by year, this is an alternative to going through a age length growth model such as the von Bertalanffy and Schnute model. CASAL2 will do a lot of interpolations of missing years for the user and across time steps. There is a condition that the measurements of length at age throughout the model years occur in the same time step.

4.8. Length-weight relationship

There are two length-weight relationships available in CASAL2. The first is the "naive" — which the relationship where the weight of an individual, regardless of length, is always 1. The second is the "basic" relationship, which is the standard cubic function of length.

4.8.0.5. `none`

$$\text{mean weight} = 1 \tag{4.34}$$

4.8.0.6. `basic`

The mean weight $\hat{w}_a$ of an individual at age a is,

$$\hat{w}_a = a\hat{l}_a^b \tag{4.35}$$

where $\hat{l}_a$ is the mean length at age a. Note that if a distribution of length-at-age is specified, then the mean weight is calculated over the distribution of lengths,

$$\hat{w}_a = (a\hat{l}_a^b)(1+cv^2)^{\frac{b(b-1)}{2}} \tag{4.36}$$

where the cv is the coefficient of variation (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). For users comparing CASAL with CASAL2, there is a small difference between the two programs. CASAL only adjusted the c.v.s `by_length` when c.v.s are used in distribution calculations (length based selectivities, length based processes and observations), and is not done in the above correction.

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).

```
@length_weight length_weight
type basic
units tonnes
a 0.00000123
b 3.132
```

4.9. Age-weight relationship

CASAL2 also allows users to input direct weight at age measurements. This is different to the method above as it uses empirical data to evaluate weight at age, rather than calculating it via the growth functions (age->length->weight).

This class represents the weight at age for categories at a point in time. They can be used in weight based derived quantities, processes and observations. An example of applying this in functionality in CASAL2 model.

```
type Data
units tonnes
table data
year 1 2 3 4 5 6 7 8 9 10
1986 0.134 0.686 1.639 2.719 3.649 4.901 6.329 6.591 7.238 7.491
1987 0.132 0.724 1.534 2.829 4.092 4.853 5.705 6.143 7.179 8.089
1988 0.122 0.641 1.533 2.641 3.796 5.054 5.652 6.356 6.95 8.857
1989 0.137 0.722 1.606 2.416 3.629 5.027 5.561 6.35 6.933 7.217
1990 0.138 0.773 1.645 2.74 3.711 4.506 5.684 6.929 7.424 7.479
end_table
```

If weight is purely defined by the weight at age functionality, you can choose not to specify the age length block in the `@categories` block.

```
@categories
format stock
names Stock
```

if a weight/biomass based derived quantity, process or observation has a `age_weight_label` subcommand then it can use the `@age_weight` class to calculate mean weight at age.

4.10. Weightless model

To model abundance (i.e., not convert the population to weight), the `@length_weight` argument is turned off by specifying the keyword `none` in the `@age_length` block, e.g.,

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

In this case any "biomass" generated by CASAL2 will actually be abundance, and care should be taken with interpretation of the output 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 to the partition allows CASAL2 to use mature elements in processes, derive mature biomass 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 their own command block (`@selectivity`), where the unique label of the selectivity is used by observations and processes to identify which selectivity to apply.

Selectivities are indexed by age, with indices from `min_age` to `max_age`. For example, for a logistic age-based selectivity with 50% selected at age 5 and 95% selected at age 7, would be defined by the `type=logistic` with parameters $a_{50} = 5$ and $a_{to95} = (7-5) = 2$. The value of the selectivity at age $x = 7$ is 0.95, and the value at age $x = 3$ is 0.05. Note, while selectivities can be length based, use with caution as more testing is needed for this functionality.

The function values for some choices of parameters, for some selectivities, can result in a 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 $(a_{50} - x)/a_{to95} > 5$ then the value of the selectivity at $x = 0$, i.e., for $a_{50} = 5$, $a_{to95} = 0.1$, 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 $(a_{50} - x)/a_{to95} = 40 > 5$).

The available selectivities are;

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

The available selectivities are described below.

4.12.1. constant

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

The constant selectivity has the estimable parameter C.

4.12.2. `knife_edge`

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{4.38}$$

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`

$$f(x) = V_x \tag{4.39}$$

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

4.12.4. `all_values_bounded`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \tag{4.40}$$

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

4.12.5. `increasing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \leq x \leq H \\ f(\alpha), & \text{if } x \geq H \end{cases} \tag{4.41}$$

The increasing ogive has non-estimable parameters L and H. The estimable parameters are π_L, $\pi_{L+1} \ldots \pi_H$ (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`

$$f(x) = \alpha / [1 + 19^{(a_{50} - x)/a_{to95}}] \tag{4.42}$$

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`

$$f(x) = \alpha - \alpha / [1 + 19^{(a_{50} - x)/a_{to95}}] \tag{4.43}$$

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

4.12.8. `logistic_producing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1)) \,/\, (1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \tag{4.44}$$

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

4.12.9. `double_normal`

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \tag{4.45}$$

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

4.12.10. `double_exponential`

$$f(x) = \begin{cases} \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \leq x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{4.46}$$

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).

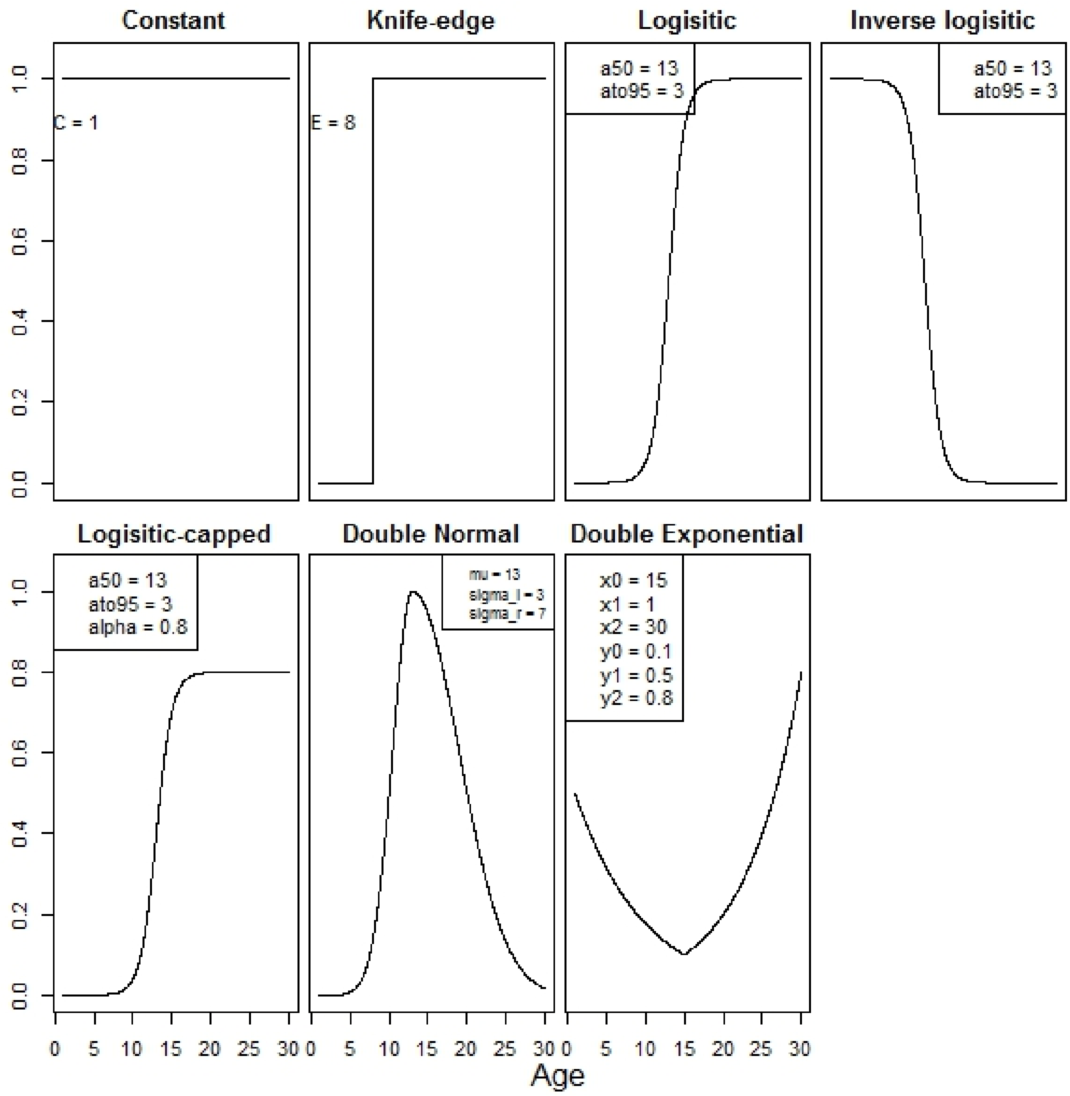

Figure 4.3: Examples of the functional forms of selectivities available in CASAL2.

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.

4.13.1. 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
```

4.13.2. 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 years' 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
```

4.13.3. lognomral

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

$$X_p = e^{\varepsilon_p - 0.5\sigma^2} \tag{4.47}$$

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
```

4.13.4. lognomral_empirical

This class applies a 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 (4.47) 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
```

4.13.5. user_defined

This class allows the use of 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 such as changing the TACC based on the current or previous state). In fisheries, this would be setting the catch based on an exploitation rate multiplied by the vulnerable biomass, where the exploitation rate is based on some rule as shown in figure 4.4.

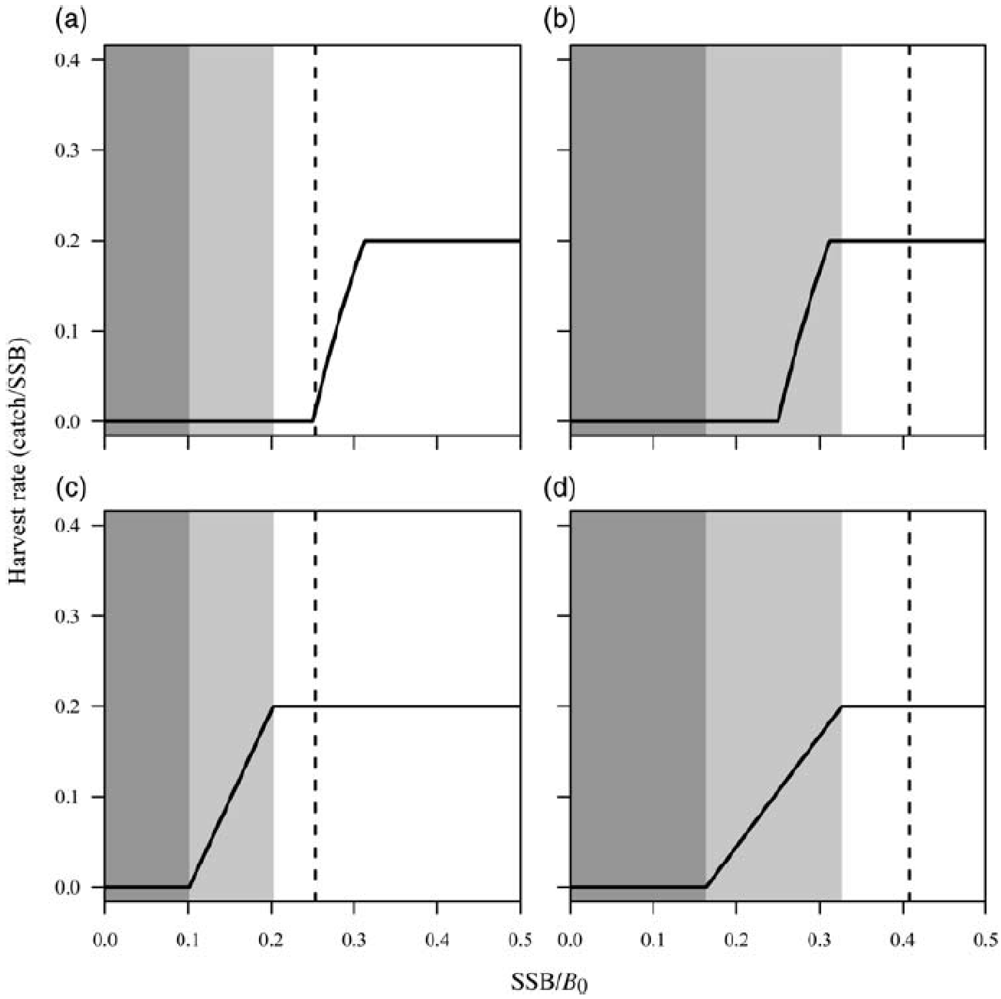

Figure 4.4: Examples of control rules based on current stock status.

```
@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 can be messy and care should be taken when writing it. In words the above equation is, if $\%B_{2014} \leq 0.1$ then set next years catch to 0.0, else if $\%B_{2014} > 0.1$ & $\%B_{2014} \leq 0.2$ then set next years catch equal to $\%B_{2014} \times SSB_{2014}$, else set next years catch to $0.2SSB_{2014}$.

4.13.6.

Projections!Catches Catches are unique in that they are known inputs in a table format. To project catches that are in a table use this example projection block,

```
# fishing process
@process Fishing
type mortality_instantaneous_retained
m 0.17*6  #0.17    #testing at old values
time_step_ratio 1
selectivities One*6   #for age based M
categories *
table catches
year FishingLine FishingPot Recreation
1900 0 0 0
1901 13.2 0 22.9
1902 26.4 0 23.5
1903 39.6 0 24
end_table

# projection block
@project future_catch
type constant
parameter process[Fishing].method_fishingpot
years 2020:2029
values 4000
```

This follows the syntax, `block_type[block_label].method_fishinglabel`. **note** the fishing label which is defined in the table needs to be lower case form in the `@projection` block.

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. For years that are not specified the parameter will default to the input, or if in an iterative state such as estimation mode, the value being trialled at that iteration. Method types 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** stochastic time varying was added for simulation purposes. It has not been tested in an estimation context. To implement hierarchical I have estimated the prior values with hyper-priors. If you were to try an implement a hierarchical model using the time varying functionality, firstly you could only do this at MCMC estimation because an MPD we would not be doing that integral which is required to obtain unbiased estimates. In an MCMC context you would be assuming a Gibbs sampler. That is every draw is from a conditional distribution and so every draw is a candidate value.

When allowing removals to have annual varying catchabilities, selectivities and other more realistic model components, simulated observations more closely model real data and associated conclusions become more useful. Another driver for implementing time-varying parameters 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 occurs. Descriptive analysis showed that when fishing was earlier relative to other years smaller fish were caught and vice versa. This can be shown in the CASAL2/Examples/2stock directory, implemented at line: `382` in the `population.csl2` file.

4.14.1. constant

Allows a parameter to have an alternative value 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, 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
```

Caution required: the actual parameter and its time varying counterpart should not both be estimated, as the time varying value will overwrite the actual parameter making the first value unidentifiable and an impossible optimization process.

4.14.2. random_walk

A random deviate is 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` block 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, we recommended an `@estimate` block is specified even though the parameter is not actually being estimated, see example syntax, 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,

$$\bar{\theta}_y = \frac{\sum_y^Y \theta_y}{Y} \tag{4.48}$$

$$\theta'_y = a\bar{\theta}_y + b\bar{\theta}_y^2 + c\bar{\theta}_y^3 \tag{4.49}$$

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.

$$\delta_y = a(E_y - \bar{E}) \tag{4.50}$$

$$\theta'_y = \theta_y + \delta_y \tag{4.51}$$

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 Projections (section 4.13), Derived quantities (section 4.6) and Reports (section 7). Examples of syntax for implementing the equation parser follow from here. For a more detailed look at the parser see `https://github.com/nickgammon/parser/blob/master/parser.cpp`

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

and can apply routine mathematical functions such as `log`, `exp`, `cos`, `sin`, `tan`, e.g.,

```
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
```

if else statements can also be linked, but the syntax becomes a little messy, e.g.,

```
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
```

Only singletons can be reference, so an equation cannot be applied to vector parameters, e.g. `process[Recruit].ycs_values{1974:1980}` can't be referenced. To see which parameters can be included in an equation parser go to the syntax section (Section 12). Any subcommand that has a `type estimable` could, in theory, be addressed in an equation parser.

With the equation parser it is difficult to catch all user configuration errors, we cannot check whether a parameter that exists in the system has been populated when the user requires it. For example, the wrong year could easily be misspecified in the case of next years (2015) removals to be based on the this years (2014) state of the population,

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

The above would be an acceptable equation but obviously will cause nonsensical results, because you are asking for a value in 2020 when you are in 2015. This is just a caution, for although the equation parser adds a great deal of flexibility, users should be careful because it is easy to misspecify models in this manner.

5. The estimation section

5.1. Role of the estimation section

The role of the estimation section is to define the tasks carried out by CASAL2:

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

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

5.2. The objective function

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

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

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

The contribution to the objective function from the likelihoods are defined in Section 6.2. 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 parameters that will be estimated (estimables) 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: ADOL-C, CPPAD, and BETADIFF. For references see Section 1.8. AD minimisers are recommended for complex models as they are on average much faster and tend to find a better minimum when 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 easily done with the `-i parameter_file.txt` in CASAL2. We recommend, when estimating any model with CASAL2 that you try smaller values for the `tolerance` parameter. We 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). This is not ideal model behaviour and that more investigative work may be required to determine what parameters are causing the behaviour. An aside note: this will also effect your covariance matrix — with different tolerances and searches you may end up with a different approximate Hessian matrix which is inverted to solve for the covariance matrix. We tested the difference this might make on MCMC results (because the covariance is incorporated into the proposal distribution). However, MCMC runs with varying tolerances and have not been found, so far, to 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; and
- 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 multi-dimensional 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, based on ADOL-C.

```
@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
```

We have found this solver to be by far the quickest solver for models that have a reasonably well defined solution, i.e., 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 nth set of values).

As CASAL2 has no post-processing capabilities. CASAL2 cannot produce MCMC convergence diagnostics (use a package such as BOA) or plot/summarize the posterior distributions of the output quantities (for example, use a general-purpose statistical or spreadsheet package such as S-Plus, **R**, or Microsoft Excel).

Bayesian methodology and MCMC are both large and complex topic. 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 briefly describes the MCMC algorithms used in CASAL2. See Section 9.3 for a description of the sequence of CASAL2 commands used in a full Bayesian analysis.

CASAL2 uses a straightforward implementation of the Metropolis-Hastings algorithm (Gelman et al., 1995, Gilks et al., 1994). The Metropolis-Hastings algorithm attempts to draw a sample from a Bayesian posterior distribution, and calculates the posterior density π, scaled by an unknown constant. The algorithm generates a 'chain' or sequence of values. Typically the beginning of the chain is discarded and every Nth element of the remainder is taken as the posterior sample. The chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

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

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

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

There is currently two options for the starting point of the Markov Chain:

- Start from the point estimate; and
- Restart a chain given a covariance matrix and sand a previous starting point.

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,
  - `@mcmc.adjustment_method=covariance`: that if the variance of the ith parameter is non-zero and less than `@mcmc.min_difference` times the difference between the parameters' lower and upper bound, then the variance is changed, without changing the associated correlations, to $k =$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 $\mathrm{var}(i)' = k$

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

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

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

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

1. You can request that the stepsize change adaptively at one or more sample numbers (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 an MCMC chain, in the case where computers get turned off and the MCMC execution was halted. This allows the ability to restart it from where it finished,

```
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 4.4.5)) or simulations (Section 6.6) with the values supplied using casal2 -i *file*.

A multivariate t distribution is used 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. Note the default is the multivariate t.

Given a posterior (sub)sample, CASAL2 can calculate a list of output quantities for each sample point (see Section 7 specifically 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 4.4.5). 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

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

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

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

3. Normal with mean μ and c.v. c

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

4. Normal with mean μ and standard deviation σ

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

5. Lognormal with mean μ and c.v. c

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

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

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

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

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

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 nonsensical, 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 within an event mortality process. 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 your model to apply less than that because that is nonsensical.

5.9. Additional Priors

Additional priors can be thought of as the inverse of penalties and for CASAL users most of the legacy `@penalty` blocks have now been migrated as `@additional_prior` blocks. They constrain or encourage parameters in user defined spaces. The types of additional priors available in CASAL2 are `vector_smoothing`, `vector_averaging`, `uniform_log`, `lognormal`, `element_difference` and `Beta`, which are 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 âĂIJsmoothedâĂİ 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`

$$-\log\left(\pi\left(p_1,p_2\right)\right)=\sum_{i=1}^{n}\left(p_{1,i}-p_{2,i}\right)^2 \tag{5.12}$$

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

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

   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 $\tau\leq 0$.

Methods available for the type `vector_average` are `l`, `k`, `m`. For a target vector parameter **X** and desired average k, the contribution to the objective score is.

- `method k`

  $-\log\left(\pi\left(p\right)\right)=\left(\bar{X}-k\right)^2$
- `method l`

  $-\log\left(\pi\left(p\right)\right)=\left(\overline{ln\left(X\right)}-k\right)^2$
- `method m`

  $-\log\left(\pi\left(p\right)\right)=\left(ln\left(\bar{X}\right)-k\right)^2$

where $\overline{ln\left(X\right)}$ is the mean of the logged values.

There are a range of parameters and derived values that users can apply additional priors to. Here are a list of non-estimated (all parameters that can be estimated can have an additional prior attached to them) parameters that you can apply additional priors on. This should be a useful guide for users who are trying to apply the equivalent old CASAL penalties to their updated CASAL2 models.

- `selectivity[Selectivity_label].values{3:8}`. This applies a selectivity to the actual selectivity value by age (in this case for ages 3-8). This is only available for certain types of selectivities (`all_values`, `all_values_bounded`, `double_exponential`). See the Hoki stock assessment for an example of applying additional priors on selectivities.
- `catchability[Catchability_label].q` this is only for catchabilities that are of type `nuisance`. Because `nuisance` q's are not free parameters to replicate legacy CASAL models with `@estimate` blocks in nuisance q's we now apply additional priors. Note, if legacy models applied uniform priors this has a null effect and you can ignore functionality when converting to CASAL2 models.

5.10. Estimate Transformations

CASAL2 has multiple methods to transform a parameter, with some methods developed for legacy purposes and others are more recent ideas. All transformations are implemented for the same reason

— to try and achieve 'better' model optimisation. It is no surprise that complex population models can have highly correlated parameters so transforming them to be orthogonal or to be in a different space is a way of trying to remove correlations, and to allow the minimiser to find a 'global' minimum quicker. 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.

There are two main methods available in CASAL2, `transform_with_jacobian` and `prior_applies_to_transform`. When using Transform-with-Jacobian the user defines priors on parameters in natural/model space (business as usual priors) but when we pass the parameter to the minimiser it gets transformed and a Jacobian is added to the objective function to account for the transformation. The second method is when users can specify bounds and prior parameters on the parameters in transformed space. Note that you cannot specify both `prior_applies_to_transform` and `transform_with_jacobian` true, CASAL2 should gracefully tell you this.

There are two ways users can apply estimate transformations. The first is within the `@estimate` block, this is for univariate (simple) transformations only. For complicated transformations you will have to specify a `@estimate_transformation` block to describe the transformation. Examples of these two implementations,

```
## simple transformation
@estimate log_R0
type lognormal
transformation log
parameter process[Recruitment].r0
transform_with_jacobian true
mu 442413
cv 0.2
lower_bound 3000
upper_bound 24154953

## Complicated
@estimate R0
type lognormal
parameter process[Recruitment].r0
mu 442413
cv 0.2
lower_bound 3000
upper_bound 24154953

@estiamte_transformation Log_R0
type log
estimate_label log_R0
transform_with_jacobian true
```

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: $supp(X) \to \mathbf{R}$ 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 multivariate 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. **Note:** if this functionality is in use be careful interpreting the covariance matrix as this will be related to the transformed variable not the variable space that you may be thinking it is in e.g. if you are estimating natural mortality (M) as $Y = M/2$ the covariance matrix will be described for Y.

```
@estimate log_R0
type lognormal
transformation log
parameter process[Recruitment].r0
transform_with_jacobian true
mu 442413
cv 0.2
lower_bound 3000
upper_bound 24154953
```

Transform without Jacobian but prior defined in transformed space

This is where users define the priors in transformed space (this class of transformations will contain functionality that was implemented in the original CASAL). If $f()$ is a transformation function defined on the support of X, then $Y = f(X)$ is a new random variable (transformed variable). In this class users specify *a priori* information with regard to $p_Y(y)$ and X can be thought of as a derived quantity. An example of this syntax,

```
@estimate log_R0
type lognormal
parameter process[Recruitment].r0
prior_applies_to_transform true
mu 13
cv 0.5
lower_bound 8
upper_bound 17
```

Transformation types

- `log` natural logarithm transformation
  `is_simple = true`
  `jacobian defined = true`

$Y = ln(X)$
$\left|\frac{\partial}{\partial y} f^{-1}(y)\right| = X^{-1}$

- `inverse`
  `is_simple = true`
  `jacobian defined = true`
  $Y = X^{-1}$
  $\left|\frac{\partial}{\partial y} f^{-1}(y)\right| = -X^{-2}$

- `sqrt` Square Root transformation
  `is_simple = true`
  `jacobian defined = true`
  $Y = \sqrt{X}$
  $\left|\frac{\partial}{\partial y} f^{-1}(y)\right| = -X^{-1.5}$

- `average_difference` Take two parameters θ_1 and θ_2 and transform to Y_1 and Y_2, where Y_1 is the average of the original parameters and Y_2 is the difference between the mean and each parameter.
  `is_simple = false`
  `jacobian defined = false`
  $Y_1 = \frac{\theta_1+\theta_2}{2}$
  $Y_2 = (Y_1 - \theta_2)2$
  Restore transformations
  $\theta_1 = Y_1 + 0.5Y_2$
  $\theta_2 = \theta_1 - 0.5Y_2$
  $\left|\frac{\partial}{\partial y} f^{-1}(y)\right|$ Hasn't been assessed (i.e it could exist)

- `log_sum` Take two parameters θ_1 and θ_2 and transform to Y_1 and Y_2, where Y_1 is the natural logorithm of the sum of θ_1 and θ_2. Y_2 describes the proportion of the sum with respect to θ_1
  `is_simple = false`
  `jacobian defined = false`
  $Y_1 = ln(\theta_1 + \theta_2)$
  $Y_2 = \theta_1/(\theta_1 + \theta_2)$
  Restore transformations
  $\theta_1 = exp(Y_1)Y_2$
  $\theta_2 = exp(Y_1)(1 - Y_2)$
  $\left|\frac{\partial}{\partial y} f^{-1}(y)\right|$ Hasn't been assessed (i.e it could exist)

- `orthogonal` Take two parameters θ_1 and θ_2 and transform to Y_1 and Y_2, where Y_1 is the multiplication of θ_1 and θ_2. Y_2 is the division of θ_1 and θ_2
  `is_simple = false`
  `jacobian defined = true`
  $Y_1 = \theta_1\theta_2$

$Y_2 = \theta_1/\theta_2$
Restore transformations
$\theta_1 = \sqrt{Y_1 Y_2}$
$\theta_2 = \sqrt{Y_1/Y_2}$
$\left|\frac{\partial}{\partial y} f^{-1}(y)\right| = 2Y_2$

- `SumToOne` Take two parameters θ_1 and θ_2 that have the constraint $\sum_{i=1}^{2} \theta_i$ and estimate onlyt θ_1 given $\theta_2 = 1 - \theta_1$
  `is_simple = false`
  `jacobian defined = false`

6. The observation section

6.1. Observations

The objective function is based on the goodness-of-fit of the model to the supplied observational data. Observations are typically supplied at an instance in time, over a group of aggregated categories. Most observations are formed from time, 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.

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 main 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` subcommand. 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 defined time step. For example, the observation type `process_abundance` is a process based observation, whereas `process_abundance` `abundance` 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 using the subcommand `time_step_proportion`. This could be useful if a survey occurs part-way through an exploitation phase, e.g when 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 (explained in the following descriptions). A list of observation `types` available with this class of observations are:

- `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 observation 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 expectations 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 a 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 in an 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. 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=false`, then CASAL2 expects a vector of proportions for each year that is $n-1$, where n is the number of lengths supplied. If `length_plus=true` then 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 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 CASAL2 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 4.5.5). Tagging events list the labels of the tags which are modelled, and define the events where fish are tagged (i.e., CASAL2 moves 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

```
## 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 format=*.Area1.*+ ## scanned categories in Area1
tagged_categories *.Area1.2011+  ## male and femaled tagged categories
detection 0.85 ## detection probability
likelihood binomial ## likelihood choice
selectivities One ## label of selectivity for tagged
tagged_selectivities 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

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 observed ($O_{y,l}$) and expected ($E_{y,l}$) values in year y and length l of this observation are calculated as followed;

$$O_{y,l} = \frac{R_{y,l}}{S_{y,l}} \tag{6.4}$$

where $R_{y,l}$ is the recaptures in year y at length l and $S_{y,l}$ are the scanned values, supplied by the user.

$$E_{y,l} = d\frac{\tilde{N}_{y,l,t} + (\tilde{N}_{y,l,t+1} - \tilde{N}_{y,l,t}) \times p}{N_{y,l,t} + (N_{y,l,t+1} - N_{y,l,t}) \times p} \tag{6.5}$$

where $\tilde{N}_{y,l,t}$ is an element in the tagged categories at the beginning of time step t and $\tilde{N}_{y,l,t+1}$ is an element in the tagged categories at the end of time step t. $N_{y,l,t}$ is the sum of the categories that were vulnerable to sampling when the obervation occured. p is the proportion of the time step that the observation was taken, d is the detection probability. For cases where there are multiple tagged categories and multiple categories that were vulnerble to sampling.

$$\tilde{N}_{y,l,t} = \sum_{j=1}^{J} N_{y,l,t,j} \tag{6.6}$$

where $j = \{1, 2, 3, ..., J\}$ are all the tagged categories, the same method is applied to the vulnerable categories to get $N_{y,l,t}$. Remember that the tagged categories should be defined in the vulnerable categories. If you think about an extreme case where we tag every individual in the population this would be divide by zero. So to constrain the expectation to be between 0-1, we need the numerator to be in the denominator.

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 4.4.2. 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_age_retained`
- `process_removals_by_age_retained_total`
- `process_removals_by_length`
- `process_removals_by_length_retained`
- `process_removals_by_length_retained_total`
- `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,

$$E[N_{a,c,m}] = N_{a,c}U_{a,m}S_{a,c,m}0.5M_{a,c} \tag{6.7}$$

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 age retained

Observations of retained and total catches by age are permitted, using the labels `process_removals_by_age_retained` and `process_removals_by_age_retained_total` respectively. Examples of two such observations are given below, with the associated process `Instantaneous_Mortality_Retained` having the form of the example in Section 4.5.3.4. First, for retained catch:

```
@observation potFishAFtotal   #test syntax get catch AF out
type process_removals_by_age_retained_total
mortality_instantaneous_process Instantaneous_Mortality_Retained
method_of_removal FishingPot
years 2005
time_step 1
categories male
### ageing_error Normal_ageing
min_age 3
max_age 15
plus_group True
table obs
2005 0.0002814574 0.0095351205 0.1661896098 0.2701718827 0.2214454177 0.1661869474 0.1107930285 0.
end_table
table error_values
2005 651
end_table
likelihood multinomial
delta 1e-11
```

and similarly, for total catch:

```
@observation potFishAFretained   #test syntax --> fits to discards not catch
type process_removals_by_age_retained
mortality_instantaneous_process Instantaneous_Mortality_Retained
method_of_removal FishingPot
years 2005
time_step 1
categories male
# ageing_error Normal_ageing
min_age 3
max_age 15
plus_group True
table obs
2005 1.650990e-10 7.566419e-07 1.771126e-03 1.962050e-01 3.192775e-01 2.413644e-01 1.609208e-01 8.
end_table
table error_values
2005 651
end_table
likelihood multinomial
delta 1e-11
```

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,

$$E[N_{a,c,m}] = N_{a,c}U_{a,m}S_{a,c,m}0.5M_{a,c} \tag{6.8}$$

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 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.

Process removals by age retained

Observations of retained and total catches by length are permitted, using the labels `process_removals_by_length_retained` and `process_removals_by_length_retained_total` respectively. Examples of two such observations are given below, with the associated process `Instantaneous_Mortality_Retained` having the form of the example in Section 4.5.3.4. First, for retained catch:

```
@observation potFishLFtotal   #test syntax get catch LF out
type process_removals_by_length_retained_total
mortality_instantaneous_process Instantaneous_Mortality_Retained
method_of_removal FishingPot
years 2005
time_step 1
categories male
length_bins 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 # for LF in catch
length_plus False
table obs
2005 0.05344612 0.06432242 0.07357780 0.08050385 0.08473451 0.08619620 0.08502982 0.08152921 0.076
end_table
table error_values
2005 651
end_table
likelihood multinomial
delta 1e-11
```

and similarly, for total catch:

```
@observation potFishLFretained   #test syntax get retained LF out
type process_removals_by_length_retained
mortality_instantaneous_process Instantaneous_Mortality_Retained
method_of_removal FishingPot
years 2005
time_step 1
categories male
length_bins 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 # for LF in catch
length_plus False
table obs
2005 0.02462879 0.03536036 0.04759163 0.06025858 0.07205340 0.08169356 0.08817806 0.09095124 0.089
end_table
table error_values
2005 651
end_table
likelihood multinomial
delta 1e-11
```

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,

$$E_a = \frac{N_a - N'_a}{N_a} \tag{6.9}$$

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;

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(Z(E_i, \delta)) \tag{6.10}$$

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

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

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;

$$-\log(L) = -\log(\Gamma\sum_i(\alpha_i)) + \sum_i \log(\Gamma(\alpha_i)) - \sum_i(\alpha_i - 1)\log(Z(O_i,\delta)) \tag{6.12}$$

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,

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

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;

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

where

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

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,

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

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 âĂŸnuisanceâĂŹ 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 âĂŤ 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,

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

where

$$\sigma_i = \sqrt{\log\left(1+c_i^2\right)} \tag{6.18}$$

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.

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

The default value of δ is 1×10^{-11}.

The normal likelihood

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

$$-\log(L) = \sum_i \left(\log(c_i E_i) + 0.5 \left(\frac{O_i - E_i}{Z(c_i E_i,\delta)} \right)^2 \right) \tag{6.20}$$

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

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

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,

$$\begin{aligned} -\log(L)' = -\sum_i [\, & \log(n_i!) - \log((n_i-m_i)!) - \log((m_i)!) + m_i \log\left(Z\left(\frac{M_i}{N_i},\delta\right)\right) \\ & + (n_i - m_i)\log\left(Z\left(1-\frac{M_i}{N_i},\delta\right)\right)\,] \end{aligned} \tag{6.22}$$

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;

$$\begin{aligned}-\log(L) = -\sum_i [\, &\log(N_i!) - \log((N_i(1-O_i))!) - \log((N_iO_i)!) + N_iO_i\log(Z(E_i,\delta)) \\ &+ N_i(1-O_i)\log(Z(1-E_i,\delta))\,]\end{aligned} \tag{6.23}$$

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

$$Z(\theta,\delta) = \begin{cases}\theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise}\end{cases} \tag{6.24}$$

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;

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{6.25}$$

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

$$Z(\theta,\delta) = \begin{cases}\theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise}\end{cases} \tag{6.26}$$

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

$$c_i' = \sqrt{c_i^2 + c_{process\_error}^2} \tag{6.27}$$

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,

$$N'_i = \frac{1}{1/N_i + 1/N_{process\_error}} \tag{6.28}$$

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	(6.29)	(6.29)	(6.31)	*	*
Lognormal	(6.32)	(6.32)	(6.36)	*	(6.37)

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,

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

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

$$\frac{\partial}{\partial q}(-log(L)) = \frac{n}{q} + \frac{1}{q^2}\sum_i \frac{O_i}{c_i^2 E_i} - \frac{1}{q^3}\sum_i \left(\frac{O_i}{c_i E_i}\right)^2 \tag{6.30}$$

hence

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4nS_2}}{2n} \tag{6.31}$$

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,

$$-log(L) = \sum_i log(\sigma_i) + 0.5\sum_i \left(\frac{log(O_i) - log(qE_i) + 0.5\sigma_i^2}{\sigma_i}\right)^2 \tag{6.32}$$

$$\frac{\partial}{\partial q}(-log(L)) = \frac{-1}{q}\sum_i \left(\frac{log(O_i/E_i) - log(q) + 0.5\sigma_i^2}{\sigma_i^2}\right) \tag{6.33}$$

$$\hat{q} = exp\frac{0.5n + S_3}{S_4} \tag{6.34}$$

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 (6.31) becomes,

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4(n+1)S_2}}{2(n+1)} \tag{6.35}$$

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 (6.34) becomes

$$\hat{q} = exp\frac{0.5n - 1 + S_3}{S_4} \tag{6.36}$$

and if the prior is lognormal,

$$\hat{q} = exp\frac{0.5n - 1.5 + log(\mu_q)/\sigma_q^2 + S_3}{S_4 + 1/\sigma_q^2} \tag{6.37}$$

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 7.14 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 15. 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 × 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*)+0.5σ^2)/σ 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 output from the model. CASAL2 does not, in general, produce any output unless specified by a valid `@report` block.

7.1. Report command block format

Reports from CASAL2 can be defined to print partition and states objects at a particular point in time, observation summaries, estimated and derived parameter values, and objective function values.

```
@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 ...
```

7.2. Report block output format

Reports from CASAL2 have a standard style (with one exception, the `output_parameters` report, see below). The standard style is that reports are prefixed with an asterix followed by a user-defined label and type of report in brackets (e.g., `*label (type)`), with the report ending with the line `*end`. For example,

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

This report block output format should make it easier for other software packages to read and process CASAL2 output. The `extract` functions in the **R** CASAL2 package use this information to identify and read CASAL2 output.

The `output_parameters` report does not print either a header or `*end` at the end of the report block. This is because the `output_parameters` report is designed to provide a single line vector of the estimated parameter values, or multiple lines for more than one set, which can be read by CASAL2 with the command `casal2 -i`. This is a specialised report for the `casal2 -o filename` command.

For estimated values in standard output use the `type=estimate_value` report.

Reports can be defined in a `@report` command block but may not be output, e.g., a report to print the partition for a year and/or timestep that does not exist, or reporting the covariance matrix when not estimation run mode.

Certain reports are associated with certain CASAL2 run modes. These reports are ignored by CASAL2 and the program will not generate any output for these reports, although they must still conform to CASAL2 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 the simulation reports will be output.

7.3. Print the partition at the end of an initialisation

This report prints the partition following the initialisation phase, which includes the numbers of individuals in each age class and category in the partition. This report will print out in run modes `-r, -e, -f`.

7.4. Print the partition

This report prints the numbers of individuals in each age class and category in the partition for each given year or given years and timestep. This report is evaluated at the end of the timestep in the given year(s). This report will print out in run modes `-r, -e, -f`.

7.5. Print the age length and length weight values

This report prints the length and weight value for each age class and category in the partition for each given year or given years and timestep. This report is evaluated at the end of the timestep in the given year(s). This report will print out in run modes `-r, -e, -f`.

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

7.6. Print a process summary

Depending on the process, different summaries are produced. These reports typically detail the type of process, its parameters and other options, and any associated details. This report will print out in run modes `-r, -e, -f`.

7.7. Print derived quantities

This report prints 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 phase. This report will print out in run modes `-r, -e, -f`.

7.8. Print the estimated parameters

This report prints a summary of the estimated parameters using the 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 run modes `-r, -e`.

7.9. Print the estimated parameters in a vector format

This report prints the estimated parameter values out as a vector. The `estimate_values` report prints the name of the parameter, followed by the value for that run. This report will print out in run modes `-r, -e`.

7.10. Print the objective function

This report prints the total objective function value, the value of all observation likelihood components, the values of all priors, and the value of any penalties that have been incurred. If an individual model run does not incur a penalty, then the penalty will not be reported. This report will print out in run modes `-r, -e, -f`.

7.11. Print the covariance matrix

This report prints the Hessian and covariance matrices if in estimation run mode and if the covariance has been requested by `@minimiser[label].covariance=true`.

7.12. Print observations, fits, and residuals

This report prints, for each category or combination of categories, the expected values, residuals (observed − expected), the error value, process error, 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 datum in the observation.

Constants in the likelihood components are often ignored in the objective function score of individual observation values. Hence, the total score from an observation equals the contribution of the objective function scores from each individual observation value plus a constant term (if applicable). In likelihood components without a constant term, the total score from an observation will equal the contribution of the objective function scores from each individual observation value.

If CASAL2 is in simulation run mode, 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
```

7.13. Print simulated observations

This report prints a complete set of observation values in the form specified by `@report[label].type=observation`, 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 run mode `-s`.

7.14. Print the ageing error misclassification matrix

This report prints the ageing error misclassification matrix used to offset observations within during model the model fitting procedure.

7.15. Print selectivities

This report prints the values of a selectivity for each age in the partition, for a given year and at then end of a given timestep.

7.16. Print the random number seed

This report prints the random number seed used by CASAL2 to initialise the generated random number sequence. Additional runs which use the same random number seed and the same model will produce identical outputs.

7.17. Print the results of an MCMC

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

7.18. Print the MCMC samples as they are calculated

This report prints the MCMC samples for each new *i*th sample as they are calculated while doing an MCMC. The output file will be appended with each new sample as it is calculated by CASAL2. This report will print out in run mode `-m`.

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

This report prints 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 appended with each new set of objective function values as it is calculated by CASAL2. This report will print out in run mode `-m`.

7.20. Print time varying parameters

This report prints all `@time_varying` blocks with the values and years in which they were specified. This report will print out in run modes `-r, -e, -m`.

```
@report time_varying_parameters
type time_varying
```

7.21. Tabular reporting format

An alternative reporting framework to the standard output is the tabular reporting format. Tabular reporting is used with multi-line `-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 specified using the `--tabular` argument (`casal2 -r --tabular -i file_name`).

Derived quantities, processes, observations, and `estimate_values` are the only report types that can be output with this format. For each input file the output will begin with the names of each column followed by a multi-line report ending with the `*end` syntax.

These tables can be read with **R** using the `CASAL2` package. An example usage is reading in files of MCMC posterior values of derived quantities, which can then 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: R,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: R,$0 \leq \text{age}_{\text{min}} \leq \text{age}_{\text{max}}$,

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

age_plus Define the oldest age or extra length midpoint ,plus group size, 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: R,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: R,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: R,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` The minimum length in each length bin
Type: non-negative integer vector
Default: true
Value: R,$0 \leq \text{length}_{\text{min}} \leq \text{length}_{\text{max}}$,

`length_plus` Specify whether there is a length plus group or not
Type: boolean
Default: true
Value: true, false

`length_plus_group` Mean length of length plus group
Type: non-negative integer
Default: 0
Value: R,$\text{length}_{\text{max}}$, length_plus_group,

`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
Value: grams, kgs or 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_initialisation_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` R,The labels of age_length objects that are assigned to categories,
Type: string vector
Default: true

`length_weight` R,The labels of the length_weight objects that are assigned to categories,
Type: string vector
Default: true

`age_weight` R,The labels of the age_weight 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: ""

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=growth_basic

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

`number_of_growth_episodes` Number of growth episodes per year
Type: non-negative integer
Default: No Default

`growth_time_steps` Time step in which each growth episode occurs
Type: string vector
Default: No Default

`cv` c.v. for the growth model
Type: constant
Default: 0.0
Lower Bound: 0.0 (inclusive)

`sigma_min` Lower bound on sigma for the growth model
Type: constant
Default: 0.0

8.5.3. `@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

8.5.4. `@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
Lower Bound: 0.0 (inclusive)

`time_step_ratio` Time step ratios for the mortality rates
Type: constant vector
Default: true
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

8.5.5. `@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: ""

8.5.6. `@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: ""

8.5.7. `@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: true

`predator_selectivities` Selectivities for predator categories
Type: string vector
Default: true

`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

8.5.8. @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: ""

8.5.9. @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: ""

8.5.10. `@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)

`time_step_ratio` Time step ratios for natural mortality
Type: constant vector
Default: true
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

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

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

`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)

`time_step_ratio` Time step ratios for natural mortality
Type: constant vector
Default: true
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

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

8.5.12. @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

8.5.13. `@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
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_initialisation_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

8.5.14. @process[label].type=recruitment_beverton_holt_with_deviations

`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
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

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

`b_max` Max bias adjustment
Type: estimable
Default: 0.85
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`last_year_with_no_bias` Last year with no bias adjustment
Type: non-negative integer
Default: false

`first_year_with_bias` First year with full bias adjustment
Type: non-negative integer
Default: false

`last_year_with_bias` Last year with full bias adjustment
Type: non-negative integer
Default: false

`first_recent_year_with_no_bias` First recent year with no bias adjustment
Type: non-negative integer
Default: false

`b0_initialisation_phase` Initialisation phase Label that b0 is from
Type: string
Default: ""

`deviation_values` Recruitment deviation values
Type: estimable vector
Default: No Default

`deviation_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

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

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

`proportions` Proportions
Type: estimable
Default: true

`length_bins` The length bins recruits are uniformly distributed over, when recruitment occurs
Type: non-negative integer vector
Default: No Default

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

8.5.16. `@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

8.5.17. `@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

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

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

`to` ategories to transition to
Type: string vector
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: ""

`selectivities`
Type: string vector
Default: No Default

`n`
Type: constant vector
Default: true

8.5.19. `@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

8.5.20. `@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

8.5.21. `@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

8.7. 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

`values`
Type: Addressable vector
Default: No Default

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
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`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

`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

8.8.1. `@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

`time_step_measurements_were_made` Time step label for which size-at-age data are provided
Type: string
Default: No Default

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

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

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

8.10. 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

`partition_type` The type of partition this selectivity will support, Defaults to same as the model
Type: string
Default: model
Allowed Values: model, age, length, hybrid

`values`
Type: Addressable vector
Default: No Default

`length_values`
Type: Addressable vector
Default: No Default

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

`same` List of parameters that are constrained to have the same value as this parameter
Type: string vector
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 estimated at the point estimate but fixed during MCMC estimation 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

9.2. 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

9.3. Markov chain Monte Carlo (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

9.4. 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

9.6. 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

9.7. 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

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 by category and 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
Lower Bound: 0.0 (inclusive)

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

`likelihood_multiplier` Likelihood score multiplier
Type: constant
Default: double

`error_value_multiplier` Error value multiplier for likelihood
Type: constant
Default: double

10.1.1. `@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

10.1.2. `@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

`age_weight_labels` R,The labels for the `@`*age_weight* block which corresponds to each category, if you want to use that weight calculation method for biomass calculations,
Type: string vector
Default: ""

10.1.3. `@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

`plus_group` 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: double

`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

10.1.4. @observation[label].type=process_removals_by_age_retained

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

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

`plus_group` 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: double

`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

10.1.5. `@observation[label].type=process_removals_by_age_retained_total`

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

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

`plus_group` 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: double

`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` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: double

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` the value of process error
Type: estimable 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_removals_by_length_retained

`length_bins` Length bins
Type: constant vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`length_plus` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: double

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` the value of process error
Type: estimable 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

10.1.8. @observation[label].type=process_removals_by_length_retained_total

`length_bins` Length bins
Type: constant vector
Default: No Default

`time_step` Time step to execute in
Type: string
Default: No Default

`length_plus` Is the last bin a plus group
Type: boolean
Default: true

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: double

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`process_errors` the value of process error
Type: estimable 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

10.1.9. @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

`plus_group` 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: double

`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: ""

10.1.10. `@observation[label].type=proportions_at_length`

`time_step` The label of time-step that the observation occurs in
Type: string
Default: No Default

`tolerance` Tolerance for rescaling proportions
Type: constant
Default: double

`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

10.1.11. `@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

`plus_group` 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

10.1.12. @observation[label].type=proportions_mature_by_age

`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

`plus_group` Use age plus group
Type: boolean
Default: true

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`ageing_error` Label of ageing error to use
Type: string
Default: ""

`total_categories` All category labels that were vulnerable to sampling at the time of this observation ,not including the categories already given,
Type: string vector
Default: true

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: double

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

`plus_group` 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

`plus_group` 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

`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: double

10.1.15. @observation[label].type=tag_recapture_by_length

`years` Years for which there are observations
Type: non-negative integer vector
Default: No Default

`length_bins` Length bins
Type: non-negative integer vector
Default: true

`length_plus` Is the last bin a plus group
Type: boolean
Default: model

`selectivities` The labels of the selectivities used for untagged categories
Type: string vector
Default: true

`tagged_selectivities` The labels of the tag category selectivities
Type: string vector
Default: No Default

`detection` Probability of detecting a recaptured individual
Type: constant
Default: No Default

`dispersion` Over-dispersion parameter ,phi,
Type: constant
Default: double

`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated
Type: constant
Default: double

10.2. Likelihoods

@likelihood *label* Define an object of type *likelihood*

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 separate 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=age_length`

`time_step` Time Step label
Type: string
Default: ""

`years` Years
Type: non-negative integer vector
Default: true

`age_length`
Type: string
Default: No Default

`category`
Type: string
Default: No 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=initialisation_partition_mean_weight`

11.1.4. `@report[label].type=partition`

`time_step` Time Step label
Type: string
Default: ""

`years` Years
Type: non-negative integer vector
Default: true

11.1.5. `@report[label].type=partition_biomass`

`time_step` Time Step label
Type: string
Default: ""

`years` Years
Type: non-negative integer vector
Default: true

11.1.6. `@report[label].type=partition_mean_weight`

`time_step` Time Step label
Type: string
Default: ""

`years` Years
Type: non-negative integer vector
Default: true

11.1.7. `@report[label].type=partition_year_cross_age_matrix`

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. Tips for setting up Casal2 model based on an existing CASAL model

For many users that are about to embark on the CASAL2 journey, firstly good luck, but secondly, most of you will be coming from a functioning CASAL model. This section focuses on transitioning from CASAL to CASAL2.

There are a range of expected reasons why CASAL2 will provide (report) different values when comparing model output to CASAL models. There are also reasons why values will differ that are not so obvious such as, reasons caused from using different compilers on different machines where over/underflow might occur. It is thought/assumed that the latter inconspicuous reasons are insignificant (or at least should be), and the 'overall' behaviour when it comes to estimation will be the same between CASAL and CASAL2. Reasons why you can expect different values reported between CASAL and CASAL2 that I have discovered so far are;

- Report rounding. There are setting with respect to std::out in CASAL that set significant figures for writing to files. So if things look truncated, there might be a very simple reason for this.
- Priors on parameters that are turned off with `upper_bound` = `lower_bound`. In both programs you can turn off the estimation of parameters by setting the bounds equal. CASAL will evaluate the prior value and add this to the objective function, you don't need to worry as this contribution is a constant so will not effect parameter inference. It may however confuse you when comparing output between the two models.
- Default values... This one seems obvious but there are a lot of switches in these programs, and even subtle things like the `delta` in CASAL2 or `r` parameter in CASAL for robustifying likelihoods can catch you out.
- order of processes. CASAL has a predefined sequence in which it executes processes with in a time step, where as CASAL2 is completely user defined.
- Length based process/observations. CASAL2 has updated the normal distribution cdf calculation (its approximated no closed form solution) with better approximations.
- Age observations currently Casal2 doesn't have the `sum_to_one` subcommand, where as CASAL does this behind the scenes. Check that this is false if you want to truly compare
- Tag Penalties. CASAL applies a penalty as sum of squares on total tagged fish in an a 'tagging episode' from the model compared to observed number tagged fish. CASAL2 applies a penalty on the transition rate by length. If you ask to apply a tags in a length bin that doesn't have fish e.g. asking to tag 2 fish of length 60-61cm when there is 0 will flag a penalty. Unsure the consequence of this during estimation.

Many of the switches between CASAL and CASAL2 are pretty similar but if there is any confusion you should go to the syntax section of this document (Sections 8). So it should be easy to get a model up and running between the two programs. One tip I have is never do an estimation run (`casal2 -e`) until you have convinced yourself that the programs give the same (keep in mind the points above) results with a **range** of parameter values using the deterministic run commands (`casal2 -r`).

The first thing you should look/investigate at when setting up a comparison between CASAL2 and CASAL is focus on the stock dynamics outputs which I call the process dynamics model (i.e. ignore observations). This is your initial age-structure, SSB's and the like. If these components differ between programs then your observations will certainly be different and thus, if you blindly did an estimation you would almost certainly get different results and possibly conclude there is a bug or something.

There are few links that you can make with certain stock outputs that will point you in a direction of processes that are misspecified. Any difference between proportions in the initial age-structure (assuming an equilibrium state) is due to M (natural mortality). For difference in absolute initial age-structure (defined as R_0 in the recruitment process) will be due to growth (`@age_length` or `@length_weight`). Most of our models are B_0 initialised so R_0 is a back calculation through the growth curve.

If you have successfully got the initial age-structure between the programs the same, then you can move on to focusing on derived quantities such as SSB's. Difference in these will generally be caused by how fishing and recruitment processes are configured. Look at things like which year class values are standardised, and choice of selectivities etc.

Once you are happy that the process dynamic model is doing the same between the two programs. I reiterate, do this with a few different set of parameter values (I suggest by using the `-i` functionality). Then you can move on to investigating the observation model. Things you want to pull out and examine are expected values between CASAL and CASAL2 assuming you have input the correct observations the difference in objective function will come from model expectations and likelihood configurations, this is where subcommands such as robustification and default values will annoy you.

Once you are satisfied that the process model and observation models are the same between CASAL and CASAL2 you can unleash an estimation run (`casal2 -e`). Now I would love to say on the first attempt everything will work out and both CASAL and CASAL2 will minimise to the same values, but from my experience they wont. If this happens to you, what I suggest you do is get the parameter values from CASAL and do a deterministic run with CASAL2 using CASAL estimated parameter values (`casal2 -r -i CASAL_mpd_pars.txt`). Then once again look at the process dynamic model, once you are satisfied inspect the observation model and see if you can identify the culprit.

The next question is how close do the model estimates and outputs have to be, before we can conclusively say the models are identical? This is an ongoing decision historically we have used subjective qualitative measures to decide whether the models are doing the same thing. A recorded comparison for the hake stock assessment can be found at Appendix B in Horn (2017).

14. Syntax conventions, examples and niceties

14.1. 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

14.1.1. 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> âĂŞ 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.200
```

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 âĂIJ2001âĂİ will be available during years 1999, 2000, 2001, 2002 and 2
# Categories with the tag value âĂIJ2005âĂİ will be available during the years 2003, 2004, 2005, 2006,
```

14.2. 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
```

Another helpful short cut using the amalgamation symbol + is if your observation wants to compare to the total combined population you can use the following format.

```
@observation CPUE
type biomass
catchability Fishq
time_step one
categories *+
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
```

if `male` and `female` are the only categories in your population, then this is the same syntax as the observation just above it.

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
```

14.3. Processes

Processes are special in how they can be defined, all throughout 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
```

14.4. 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 constraints
@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 ##
length_bins 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230
     240 250

#CATEGORIES
@categories
format stock ## Single sex and area population
names HAK4
age_lengths age_size

@initialisation_phase Equilibrium_state
type derived
```

```
## Define the processes in the Annual Cycle
## This is a list of labels that correspond to a process
@time_step Sep_Feb
processes Recruitment Instantaneous_Mortality

@time_step Mar_May
processes Instantaneous_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 the example of the output.

15. Post-processing output using R

R (`https://www.r-project.org/`) is the main application used to process and visualise output from a CASAL2 model. **R** is free and can be downloaded from `https://cran.r-project.org/`. Once you have installed **R** you can install the `casal2` **R** package from the file (`casal2_1.0.tar.gz`) which is part of the CASAL2 download.

The CASAL2 **R** package has functionality to parse CASAL2 output into a list. It also has diagnostic, plotting, and summarising functions.

There are three types of output that CASAL2 can produce, depending on the type of analysis run. These outputs are: Standard, MCMC, and Derived Quantity.

The Standard outputs are the reports that are produced in most CASAL2 run modes, with the exception of `-s` and `-m`. The Standard output can be split into two additional categories, a single parameter run (`casal2 -r`) or a multi-parameter run (`casal2 -r -i many_pars.out`), or running in projection mode (`-f 1`). The Standard outputs can be read into **R** using the `extract.mpd()` function.

The second type of output is generated when doing an MCMC analysis (`casal2 -m`), which can generate two files, `mcmc_objective.out` and `mcmc_samples.out`. The MCMC outputs can be used to summarise convergence properties or chain behaviour, and can also be used to view marginal posteriors and quantify parameter uncertainty.

The third output type is the Derived Quantity outputs, also referred to as tabular output. The Derived Quantity output can be generated after an MCMC analysis is done, to produce the marginal posteriors for derived quantities. A commonly reported derived quantity in fisheries stock assessment modelling is the time series of spawning stock biomass. To get the posterior distributions for these derived quantities use the `--tabular` flag (e.g., `casal2 -r -i mcmc_samples.out --tabular > Tabular_report.out`). This output can then be read into **R** using the `extract.tabular()` function.

CASAL2's reported output is written so that each `@report` will start with a '*' and end with '*end'. This format can be used as the basis to construct functions that read CASAL2 output to identify and read individual reports for post-processing.

The CASAL2 **R** `extract()` functions differ by how the expected output is structured and they each create a different `casal2` object. The `summary()` and `plot()` functions will generate different plots for the different `casal2` objects. Objects produced by the `extract()` function can be queried with `class(object).`

The list of `casal2` **R** functions include:

- `extract.mpd()`, which parses the CASAL2 default output into a list
- `extract.mcmc()`, which parses the CASAL2 MCMC output into a list
- `extract.tabular()`, which parses the CASAL2 tabular output into a list
- `extract.parameters()`, which parses the CASAL2 parameter files into a list
- `generate.starting.pars()`, which reads in a file that contains the `@estimate` blocks and generates 'N' starting values to test convergence (???)
- `burn.in.tabular()`, which omits the first 'N' rows from a `casal2TAB` object
- `plot.derived_quantities()`, which plots the derived quantities
- `plot.selectivities()`, which plots the selectivities

- `plot.ycs()`, which plots the true YCS strengths
- `plot.pressure()`, which plots the fishing pressures
- `summary()`, which summarises a model run
- `extract.csl2.file()`, which reads a CASAL2 .csl2 (configuration) file into a list
- `write.csl2.file()`, which writes a CASAL2 .csl2 (configuration) file from (???)
- `ReadSimulatedData()`, which parses CASAL2 output from a `casal2 -s` run
- `Method.TA1.8()`, which returns a weighting factor for age or length composition data. See Francis (2011) for more detail.
- `apply.dataweighting.to.csl2()`, which parses a CASAL2 .csl2 (configuration) file that contains `@observation` blocks, applies a weighting factor to an age or length composition data set, and generates a new `.csl2` file with modified effective sample size values

The required and optional arguments for these functions can be queried after loading the CASAL2 **R** library with `library(casal2)` and using the standard **R** help syntax `?` (e.g., `?param.profile()`). Many of the help files have example code and data to demonstrate function syntax.

Standard diagnostic functions and plots for model output

TODO (functionality description)

`plot.derived_quantity()`

When comparing model output either: different parameters for the same model structure are being compared (Situation 1), or outputs from multiple model structures are being compared (Situation 2). These functions can be useful for both comparison types.

- `plot.selectivities()`
- `plot.pressure()`
- `plot.fit()`
- `plot.ycs()`

Data weighting

An important component of fisheries stock assessment modelling is addressing data conflicts through the use of data weighting. There are a range of methods that can be used (Francis (2011)). The CASAL2 **R** function is `Method.TA1.8()`. An additional function `apply.dataweighting.to.csl2()` automatically applies a weighting factor to a specific age or length composition data in an `@observation` block, and generates a new `.csl2` file with modified effective sample size values.

```
library(casal2)

## read in the reported output from a "casal2 -e" run
## ensure there is a @report block for the observation of interest.
mpd <- extract.mpd(file = "estimate.log")

## calculate weighting factor from Francis method
WeightingFactor <- Method.TA1.8(model = mpd, observation_labels = "chatTANage")

## Apply the weighting factor to the block in the Observation.csl2 file
```

```
## this call generates a new file (Observation.csl2.0) with the re-weighted effective sample
    sizes
apply.dataweighting.to.csl2(weighting_factor = WeightingFactor,
                            Observation_csl2_file = "Observations.csl2",
                            Observation_label = "chatTANage",
                            Observation_out_filename = "Observation.csl2.0")
```

Automating the data weighting process:

```
library(casal2)

mpd <- extract.mpd(file = "estimate.log")

ModelFactor <- Method.TA1.8(mpd, observation_labels = c("ObserverProportionsAtAge"))

## make a back-up copy of the file Observation.csl2 before running this section

while(abs(ModelFactor - 1) > 0.01) {
  shell("betadiff & casal2 -e > estimate.log 2> log.out")

  new_mpd <- extract.mpd(file = "estimate.log")

  ModelFactor <- Method.TA1.8(new_mpd, observation_labels = c("ObserverProportionsAtAge"))

  apply.dataweighting.to.csl2(weighting_factor = ModelFactor,
                              Observation_csl2_file = "Observation.csl2",
                              Observation_out_filename = "Observation.csl2",
                              Observation_label = c("ObserverProportionsAtAge"))
  print(ModelFactor)
}
```

Troubleshooting the `casal2` R package

If you get this error when using one of the `extract()` functions

```
Read 1 item
Warning messages:
1: In scan(filename, what = "", sep = "\n", fileEncoding = fileEncoding) :
embedded nul(s) found in input
2: In extract.mpd(file = "results.txt", fileEncoding = "") :
File is empty, no reports found
```

You may be able to resolve this issue by using an alternative UTF format by specifying this format with the `fileEncoding` parameter

```
MyOutput <- extract.mpd(file = "Estimate.log", path = getwd(), fileEncoding = "UTF-16LE")
```

16. Troubleshooting

This section is to aid users in debugging models. If you cannot resolve an issue using these guidelines then please contact the development team. To report an issue please follow the format described in Section 16.2.1.

Most user errors should be well documented and CASAL2 should produce informative error messages. There are runtime options that users can enable to attempt to resolve or at least isolate an error or bug, including different levels of logging.

16.1. Logging

CASAL2's internal logging system can be invoked at the command line with argument `-loglevel` followed by one of these options: `trace`, `finest`, `fine`, `medium`.

An example of logging with trace level output:

- On Windows: `casal2 -r -loglevel trace > output.log 2> log.out`
- On Linux: `casal2 -r -loglevel trace > output.log 2&> log.out`

This argument will output CASAL2's reports to the file "output.log", and the "`2>`" or "`2&>`" syntax will print the error logged information to the file "log.out". You should be able to see where CASAL2 is exiting by going to the end of the "log.out" file.

The optimal level of logging will depend on what run mode you are using and the granularity of information that you would like to see. There is an ordering in the options, with `medium` being the most coarse, and `trace` being the finest level, with `fine` and `finest` in between. We suggest that if you are running CASAL2 in an iterative state such as for estimation (`casal2 -e`) or MCMC you use `medium` level. This is because the logging can print a lot of information for a single model run, so an estimation which could comprise thousands of model runs can produce very large text files with the finer logging option specified. For a single iteration run such as `casal2 -r` each of the logging options can be useful during different phases of model development.

You can see how CASAL2 creates these reports by looking in the ".cpp" files in the Observation or Processes source code subdirectories and see code such as in `Model/Model.cpp`,

```
LOG_FINE() « "Model:  State change to Execute";
```

16.2. Reporting errors

If you find a bug or error in CASAL2, please submit an issue in the GitHub repository at `https://github.com/NIWAFisheriesModelling/CASAL2/issues`.

Please follow the guidelines below so that the bug or error can be reproduced. It is helpful to be as detailed and specific as possible when describing the observed behavior as well as the expected behaviour.

16.2.1. Guidelines for reporting an error with CASAL2

1. Ensure you are using the most recent version of CASAL2, as the bug or error you are having may have already been resolved.

2. Provide the version of CASAL2 you are using, e.g., "CASAL2 v2019-12-12 (rev. 8ca79ad)". The version is output by CASAL2 with the command `casal2 -v`.
3. Provide the operating system you are using, e.g., "IBM-PC Intel CPU with Microsoft Windows 10 Enterprise".
4. Provide a brief description of the problem, e.g., "a segmentation fault was produced".
5. If the problem is reproducible, please describe in detail the steps required to cause it, and include the CASAL2 configuration files, other input files, and any output files generated. Specify the *exact* command line arguments that were used, e.g., "Using the command `casal2 -e -q` produced a segmentation fault. The input configuration files are attached."
6. If the problem is not reproducible (it happened only once, or occasionally for no apparent reason), please describe in detail the circumstances in which it occurred and the behaviour observed, e.g., "CASAL2 crashed, but I have not been able to reproduce the issue. It seemed to be related to a local network crash but I cannot be sure."
7. If the problem produced any error messages, please give the *exact* text displayed, e.g., "`segmentation fault (core dumped)`".
8. Attach all relevant input and output files so that the problem can be reproduced; these files can be compressed into a single file e.g., a zip file, and uploaded to GitHub.

17. 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”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence

you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

18. Acknowledgements

We thank the developers of CASAL (Bull et al., 2012) for their ideas that led to the development of 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.

TODO rewrite this 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.

19. 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.

Ian Doonan, Kath Large, Alistair Dunn, Scott Rasmussen, Craig Marsh, and Sophie Mormede. Casal2: New zealand's integrated population modelling tool. *Fisheries Research*, 183:498–505, 2016.

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.

RIC Chris Francis. Data weighting in statistical fisheries stock assessment models. *Canadian Journal of Fisheries and Aquatic Sciences*, 68(6):1124–1138, 2011.

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.

PL Horn. Stock assessment of hake (*Merluccius australis*) on the Chatham Rise (HAK 4) and off the west coast of South Island (HAK 7) for the 2016–17 fishing year. *New Zealand Fisheries Assessment Report*, 47, 2017.

J Jurado-Molina, P A Livingston, and J N Ianelli. Incorporating predation interactions in a statistical catch-at-age model for a predator-prey system in the eastern bering sea. *Canadian Journal of Fisheries and Aquatic Sciences*, 62:1865–1873, 2005.

Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10: 1755–1758, 2009.

P.M Mace and I.J Doonan. A generalised bioeconomic simulation model for fish population dynamics. *New Zealand Fisheries Assessment Report*, 4, 1988.

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.

Richard D Methot Jr and Ian G Taylor. Adjusting for bias due to variability of estimated recruitments in fishery assessment models. *Canadian Journal of Fisheries and Aquatic Sciences*, 68(10):1744–1760, 2011.

M I Michaelis and L Menten. Die kinetik der invertinwirkung. *Biochemische Zeitschrift*, 49:333–369, 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.

20. Index

21. Quick reference

@additional_prior *label* Define an object of type *additional_prior*

`parameter` Name of the parameter to generate additional prior on
`label` Label for teh additional prior
`type` Type of additional prior
@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.
`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=data

`external_gaps`
`internal_gaps`
`length_weight` The label from an associated length-weight block
`time_step_measurements_were_made` Time step label for which size-at-age data are provided

@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

@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
@catchability *label* Define an object of type *catchability*

`label` Label of the catchability
`type` Type of catchability
@categories *label* Define an object of type *categories*

`format` The format that the category names adhere too
`names` The names of the categories to be used in the model
`years` The years that individual categories will be active for. This overrides the model values
`age_lengths` R,The labels of age_length objects that are assigned to categories,
`length_weight` R,The labels of the length_weight objects that are assigned to categories,
`age_weight` R,The labels of the age_weight objects that are assigned to categories,
@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
`values`

@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
`same` List of parameters that are constrained to have the same value as this parameter
`estimation_phase` The first estimation phase to allow this to be estimated
`mcmc` Indicates if this parameter is estimated at the point estimate but fixed during MCMC estimation 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_transformation` *`label`* Define an object of type *estimate_transformation*

`label` Label for the transformation block
`type` Type of transformation
`transform_with_jacobian` Apply jacobian during transformation
`@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_initialisation_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
`@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
`@likelihood` *`label`* Define an object of type *likelihood*

`@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,
`@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
`@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 or extra length midpoint ,plus group size, 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` The minimum length in each length bin
`length_plus` Specify whether there is a length plus group or not
`length_plus_group` Mean length of length plus group
`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=abundance`

`selectivities` Labels of the selectivities
`time_step` The label of time-step that the observation occurs in

`@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
`age_weight_labels` R,The labels for the `@`*age_weight* block which corresponds to each category, if you want to use that weight calculation method for biomass calculations,

@observation[label].type=process_removals_by_age

`min_age` Minimum age
`max_age` Maximum age
`plus_group` 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_age_retained

`min_age` Minimum age
`max_age` Maximum age
`plus_group` 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_age_retained_total

`min_age` Minimum age
`max_age` Maximum age
`plus_group` 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` 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_removals_by_length_retained

`length_bins` Length bins
`time_step` Time step to execute in
`length_plus` 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_removals_by_length_retained_total

`length_bins` Length bins
`time_step` Time step to execute in
`length_plus` 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=proportions_at_age

`min_age` Minimum age
`max_age` Maximum age
`plus_group` 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

@observation[label].type=proportions_at_length

`time_step` The label of time-step that the observation occurs in
`tolerance` Tolerance for rescaling proportions
`years` Years for which there are observations
`selectivities` The labels of the selectivities
`process_errors` Process error

@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
`plus_group` Use age plus group
`years` Years for which there are observations
`selectivities` The labels of the selectivities
`categories2` Target Categories
`selectivities2` Target Selectivities

@observation[label].type=proportions_mature_by_age

`min_age` Minimum age
`max_age` Maximum age
`time_step` The label of time-step that the observation occurs in
`plus_group` Use age plus group
`years` Years for which there are observations
`ageing_error` Label of ageing error to use
`total_categories` All category labels that were vulnerable to sampling at the time of this observation ,not including the categories already given,
`time_step_proportion` Proportion through the mortality block of the time step when the observation is evaluated

@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
`plus_group` 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
`plus_group` 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
`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

`years` Years for which there are observations
`length_bins` Length bins
`length_plus` Is the last bin a plus group
`selectivities` The labels of the selectivities used for untagged categories
`tagged_selectivities` The labels of the tag category selectivities
`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
@process *`label`* Define an object of type *process*

`label` The label of the process
`type` The type of process

@process[label].type=ageing

`categories` The labels of the categories

@process[label].type=growth_basic

`categories` The labels of the categories
`number_of_growth_episodes` Number of growth episodes per year
`growth_time_steps` Time step in which each growth episode occurs
`cv` c.v. for the growth model
`sigma_min` Lower bound on sigma for the growth model

@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

@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_instantaneous_retained

`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=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_initialisation_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_beverton_holt_with_deviations

`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,
`sigma_r` Sigma r
`b_max` Max bias adjustment
`last_year_with_no_bias` Last year with no bias adjustment
`first_year_with_bias` First year with full bias adjustment
`last_year_with_bias` Last year with full bias adjustment
`first_recent_year_with_no_bias` First recent year with no bias adjustment
`b0_initialisation_phase` Initialisation phase Label that b0 is from
`deviation_values` Recruitment deviation values
`deviation_years` Recruitment years. A vector of years that relates to the year of the spawning event that created this cohort

@process[label].type=recruitment_constant

`categories` Categories
`proportions` Proportions
`length_bins` The length bins recruits are uniformly distributed over, when recruitment occurs
`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` ategories to transition to
`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
@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 separate file
`write_mode` The write mode

@report[label].type=age_length

`time_step` Time Step label
`years` Years
`age_length`
`category`

@report[label].type=ageing_error_matrix

`ageing_error` Ageing Error label

@report[label].type=initialisation_partition_mean_weight

@report[label].type=partition

`time_step` Time Step label
`years` Years

@report[label].type=partition_biomass

`time_step` Time Step label
`years` Years

@report[label].type=partition_mean_weight

`time_step` Time Step label
`years` Years

@report[label].type=partition_year_cross_age_matrix

@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
`partition_type` The type of partition this selectivity will support, Defaults to same as the model
`values`
`length_values`

@simulate *label* Define an object of type *simulate*

`label` Label
`type` Type
`years` Years to recalculate the values
`parameter` Parameter to Simulate

@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

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

`label` The time-varying label
`type` The time-varying type
`years` Years in which to vary the values
`parameter` The name of the parameter to time vary

Appendices

A. Investigating two options for YCS prior distribution formulations

There are two common ways of parameterising the log-Normal prior distribution of year class strength (YCS) when fitting models in fisheries.

Let YCS_y represent the YCS for year y. The two parameterisations used are:

1. Option 1: $YCS_y \sim LN(\mu, \sigma_R^2)$, with μ chosen so that $E(YCS_y) = 1$.
2. Option 2: $YCS_y = e^{\varepsilon_y - \frac{1}{2}\sigma_R^2}$, where $\varepsilon_y \sim N(0, \sigma_R^2)$.

To check whether the two representations are equivalent, we will determine, in each case the density function of YCS_y on the log-scale.

Note that, in general, if $Y \sim LN(\mu, \sigma_R^2)$ (ie random variable Y follows a log-Normal distribution with parameters μ and σ_R^2), then the expectation, $E(Y)$, and variance, $Var(Y)$, of Y are given by

$$E(Y) = e^{\mu + \frac{1}{2}\sigma_R^2},$$

and

$$Var(Y) = \left[e^{\sigma_R^2} - 1\right] e^{2\mu + \sigma_R^2}.$$

The log-Normal distribution can be expressed on the log scale as follows:

$$\log Y \sim \text{Normal}(\mu, \sigma_R^2).$$

Option 1: $YCS_y \sim LN(\mu, \sigma_R^2)$, with $E(YCS_y) = 1$

Setting $E(YCS_y) = 1$ implies

$$\begin{aligned} e^{\mu + \frac{1}{2}\sigma_R^2} &= 1 \\ \Rightarrow \mu + \frac{1}{2}\sigma_R^2 &= \log 1 \\ \Rightarrow \mu &= -\frac{1}{2}\sigma_R^2 \end{aligned} \tag{A.1}$$

and

$$\begin{aligned} Var(YCS_y) &= \left[e^{\sigma_R^2} - 1\right] e^{2\mu + \sigma_R^2} \\ &= \left[e^{\sigma_R^2} - 1\right] e^{2(-\frac{1}{2}\sigma_R^2) + \sigma_R^2} \\ &= \left[e^{\sigma_R^2} - 1\right] e^{0} \\ &= e^{\sigma_R^2} - 1. \end{aligned} \tag{A.2}$$

So, on the log scale we have:

$$\log YCS_y \sim N\left(-\frac{1}{2}\sigma_R^2, \sigma_R^2\right).$$

Option 2: $YCS_y = e^{\varepsilon_y - \frac{1}{2}\sigma_R^2}$, **where** $\varepsilon_y \sim N(0, \sigma_R^2)$

In this case, $YCS_y = e^{\varepsilon_y - \frac{1}{2}\sigma_R^2}$ implies

$$\log YCS_y = \varepsilon_y - \frac{1}{2}\sigma_R^2$$

and

$$E(\log YCS_y) = E\left(\varepsilon_y - \frac{1}{2}\sigma_R^2\right) = -\frac{1}{2}\sigma_R^2,$$

since $E(\varepsilon_y) = 0$. We also have:

$$\begin{aligned} Var(\log(YCS_y)) &= Var\left(\varepsilon_y - \frac{1}{2}\sigma_R^2\right) \\ &= Var(\varepsilon_y) \\ &= \sigma_R^2 \end{aligned} \tag{A.3}$$

Therefore

$$\log YCS_y \sim N\left(-\frac{1}{2}\sigma_R^2, \sigma_R^2\right).$$

Therefore the two parameterisations result in exactly the same distribution for YCS_y values and should give the same results **if expressed correctly** in MCMC algorithms.

To illustrate that these two distributions are exactly the same, we first use simulations to show that we get the same YCS_y values when generating sequences from these two formulations. One is generated directly from the log-Normal distribution, while the other is obtained by transforming a Normal random variable.

Investigating prior specification

Given the two different representations, the question is how should their prior distribution contributions to the negative log-posterior be specified?

Prior based on Option 1

For Option 1, this is straight-forward. The YCS's are generated from a log-Normal distribution, so the contribution to the log posterior is based on the log-Normal density function. That is, if we let $Y = YCS_y$, then the density function of Y is given by

$$f(y) = \frac{1}{y\sigma_R\sqrt{2\pi}} e^{-\frac{1}{2\sigma_R^2}(\log y - \mu)^2}.$$

Since $\mu = -\frac{1}{2}\sigma_R^2$ as shown in equation A.1, we have

$$-\log f(y) = \log y + \log \sigma_R + \frac{1}{2}\log 2\pi + \frac{1}{2\sigma_R^2}\left(\log y - (-\frac{1}{2}\sigma_R^2)\right)^2. \tag{A.4}$$

Prior based on Option 2

For Option 2, I will look at two ways used to specify the prior and say which one is correct.

Prior 2 - Normal distribution for ε_y.

In this approach, we have $YCS_y = e^{\varepsilon_y - \frac{1}{2}\sigma_R^2}$, where $\varepsilon_y \sim N(0, \sigma_R^2)$.

What is sometimes done is to then express the contribution to the negative log-posterior using $-\log f(\varepsilon_y)$,

where

$$f(\varepsilon_y) = \frac{1}{\sigma_R\sqrt{2\pi}} e^{-\frac{1}{2\sigma_R^2}\varepsilon_y^2},$$

and therefore

$$-\log f(\varepsilon_y) = \log \sigma_R + \frac{1}{2}\log 2\pi + \frac{1}{2\sigma_R^2}\varepsilon_y^2. \tag{A.5}$$

However, this contribution is based on the density function for ε_y and not for Y. Such an approach is incorrect. The two contributions are different as seen in equations A.4 and A.5, and as shown below.

So what does this mean in practice?

The overall model is based on YCS_y, rather than ε_y. Using a negative log-posterior based on $f(\varepsilon_y)$ gives incorrect weights to the YCS_y values in the model, meaning that in MCMC steps, acceptance probabilities will be incorrect. When using the specification based on ε_y, the correct approach is to use variable transformation methods to obtain probability densities for the YCS_y values. These density values based on $f(Y)$ are the ones to use in the negative log-posterior.

Prior 3 - Variable transformation to obtain $f(Y)$ based on $f(\varepsilon_y)$

Given $Y = e^{\varepsilon_y - \frac{1}{2}\sigma_R^2}$, where $\varepsilon_y \sim N(0, \sigma_R^2)$, we need to find $g(y)$, the distribution of the transformed variable YCS_y.

Variable transformation theory tells us that:

$$g(y) = f(s(y)) \left| \frac{ds(y)}{dy} \right|,$$

where $s(y) = \varepsilon_y(y)$ is the result of the conversion from Y to ε_y, and $\left|\frac{ds(y)}{dy}\right|$ is the Jacobian of the transformation.

We find $s(y)$ by expressing ε_y as a function of y:

$$\begin{aligned} y &= e^{\varepsilon_y - \frac{1}{2}\sigma_R^2} \\ \Rightarrow \log y &= \varepsilon_y - \frac{1}{2}\sigma_R^2 \\ \Rightarrow \varepsilon_y &= \log y + \frac{1}{2}\sigma_R^2. \end{aligned} \tag{A.6}$$

Therefore

$$\frac{ds(y)}{dy} = \frac{d\varepsilon_y(y)}{dy} = \frac{d}{dy}\left(\log y + \frac{1}{2}\sigma_R^2\right) = \frac{1}{y}.$$

Then

$$\begin{aligned} g(y) &= \frac{1}{\sigma_R\sqrt{2\pi}} e^{-\frac{1}{2\sigma_R^2}\left[\log y + \frac{1}{2}\sigma_R^2\right]^2} \cdot \left|\frac{1}{y}\right| \\ &= \frac{1}{y\sigma_R\sqrt{2\pi}} \exp\left\{-\frac{[\log y - (-\frac{1}{2}\sigma_R^2)]^2}{2\sigma_R^2}\right\}. \end{aligned} \tag{A.7}$$

This is the density function of a logNormal distribution with parameters $\mu = -\frac{1}{2}\sigma_R^2$ and σ_R^2, that is

$$Y \sim LN\left(-\frac{1}{2}\sigma_R^2, \sigma_R^2\right),$$

and from equation A.7 we see that

$$-\log g(y) = \log y + \log \sigma_R + \frac{1}{2}\log 2\pi + \frac{1}{2\sigma_R^2}\left(\log y - (-\frac{1}{2}\sigma_R^2)\right)^2. \tag{A.8}$$

which is exactly the same expression as that for $-\log f(y)$ in equation A.4.

Therefore if used correctly, Option 2 parameterisation results in the same contribution, $-\log g(y)$, to the negative log posterior as Option 1 ($-\log f(y)$.

Note that $-\log f(y) = -\log f(\varepsilon_y)$ only for $Y = 1$.

I'll carry out a brief simulation exercise to illustrate the point. I will use the density functions derived above, rather than existing R functions and calculate $-\log f(y) = -\log g(y)$ and $-\log f(\varepsilon_y)$ for a sequence of YCS_y values.

The simulation results indicate that $-\log f(y) = -\log g(y) = -\log f(\varepsilon_y)$ only for $Y = 1$. For other values of Y, the size of the difference between $-\log f(y)$ and $-\log f(\varepsilon_y)$ is given by $|\log y|$ and does not depend on σ_R^2. Therefore, the differences increase in size as the YCS value diverges further from 1.

What is the implication of this?

Incorrect use of the prior based on ε_y (ie using $-\log f(\varepsilon_y)$ in place of $-\log g(y)$) results in prior contributions to the negative log-posterior that are lower or higher by $\log YCS_y$ than what they should be.