Casal2 User Manual for Length-Based Models

Casal2 Development Team

NIWA Technical Report 139
ISSN 1174-2631
2024

User manual for length-based models with Casal2 v24.10 (2024-10-07).
https://github.com/alistairdunn1/CASAL2:Development.

Citation: Casal2 Development Team (2024). Casal2 user manual for length-based models with v24.10 (2024-10-07). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 139*. 239 p. (Using source code from https://github.com/alistairdunn1/CASAL2:Development)

Contents

1 Introduction

1.1 About Casal2

Casal2 is an open-source integrated statistical catch-at-age or catch-at-length assessment tool for modelling the population dynamics of marine populations. Casal2 is designed for quantitative assessments of marine populations, including fish, invertebrates, marine mammals and seabirds.

Casal2 implements generalised age (Casal2 for age-based models) or length structured (Casal2 for length-based models) population models that allows for a great deal of choice in specifying the population dynamics, parameters and which parameters should be estimated, and the model outputs. Casal2 is designed for flexibility. It allows implementation of age or length structured models from single species or stocks, to multiple species or stocks, using user-defined categories such as area, sex, and maturity stage. The categories are generic, are not predefined, and are easily specified. Casal2 models can be used for a single population with a single anthropogenic event (i.e., a single fish stock with a single fishery), or for multiple species and populations, areas, and/or anthropogenic or exploitation methods, and can include predator-prey interactions.

In Casal2 the processes and observations that occur over each year are defined by the user. Processes include recruitment, natural mortality, and anthropogenic mortality. Observations used to fit the models can be from many different sources, including removals-at-size or -age (e.g., a fishery), research survey or other biomass indices, and mark-recapture data. Model parameters can be estimated using penalised maximum likelihood or Bayesian methods.

As well as the point estimates of the parameters, Casal2 can calculate the likelihood or posterior distribution profiles for estimated parameters, and can generate Bayesian posterior distributions using Markov chain Monte Carlo methods. Casal2 can project the population status into the future using either deterministic or stochastic population dynamics. Casal2 can also simulate observations from a given model for both existing and potential observations.

The Casal2 user manual has been split into two separate manuals for the age-based functionality and for the length-based functionality. These two manuals contain many common components but differ in processes and observations. This manual is for length-based models. For age-based models, see the Casal2 user manual for age-based models.

1.2 Citing Casal2

The reference for this document is: Casal2 Development Team (2024). Casal2 user manual for length-based models with v24.10 (2024-10-07). National Institute of Water & Atmospheric Research Ltd. *NIWA Technical Report 139*. 239 p. (Using source code from https://github.com/alistairdunn1/CASAL2:Development)

The peer-reviewed journal article reference for Casal2 is Doonan et al. (2016).

Casal2 has also been simulation tested using simulated data from CASAL to validate results (Dunn et al., 2022).

1.3 Casal2 Contributors

The Casal2 project is maintained by the Casal2 Development Team. Casal2 was initiated by Alistair Dunn. The software architect and lead author of the software code was Scott Rasmussen. Contributors to the development of Casal2 are Scott Rasmussen, Alistair Dunn, Ian Doonan, Craig Marsh, Teresa A'mar, Kath Large, Sophie Mormede, Samik Datta, Matt Dunn, Jingjing Zhang, Marco Kienzle, and Arnaud Grüss.

The development of Casal2 was funded by National Institute of Water & Atmospheric Research

Ltd. (NIWA), with additional funding from the New Zealand Ministry for Primary Industries. More recent developments of this version were funded by Ocean Environmental Ltd.

1.4 Software license

1.5 Where to get Casal2

This version of Casal2 is hosted on GitHub, and can be found at `https://github.com/alistairdunn1/CASAL2`.

There are installation packages available for Linux and Microsoft Windows. Release versions of the package includes the Casal2 binary, the Casal2 **R** library, the Casal2 User Manual and associated documentation, example models, and other information. The installation packages for older versions of Casal2 can be downloaded at `https://github.com/NIWAFisheriesModelling/CASAL2/releases`. For more recent versions of the compiled packages, please contact the Casal2 Development Team.

See `https://www.niwa.co.nz/` or `https://github.com/alistairdunn1/CASAL2` for more information about Casal2.

1.6 System requirements

Casal2 is available for most x86 compatible machines running 64-bit Linux and Microsoft Windows operating systems. Casal2 has not been compiled nor tested on MacOS.

Several of Casal2's tasks are 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 even take several days to complete an MCMC estimate.

Output files have the potential to be large, and the output from developing a model, sensitivity analyses, and running multiple MCMC chains can take up significant 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 larger models, several gigabytes of RAM and disk space may occasionally be required.

1.7 Necessary files

For both 64-bit Linux and Microsoft Windows, we recommend using the zip files available on `https://github.com/alistairdunn1/CASAL2` for Microsoft Windows and Linux; older Linux systems can use Casal2.tar.gz. Running Casal2 on a system requires the main binary ("casal2.exe" on Windows, or "casal2" on Linux) and the associated dynamically linked libraries (DLL) for Windows or shared objects (.so) for Linux, and cannot be (easily) run by copying the binary to a working directory. Casal2 is not available for 32-bit operating systems or MacOS.

Casal2 does not post-process model output. Casal2 writes all output to text files — either to standard output or directed to files. A package that allows tabulation and graphing of model outputs is recommended. Software such as **R** (R Core Team, 2014) is recommended. The Casal2 **R** package

is provided for extracting Casal2 output from reports and output files into **R** (see Section 17). A separate **R** package, r4Casal2, is also available on GitHub and provides some examples of diagnostic and plot functionality.

1.8 Getting help

Casal2 is distributed as unsupported software. Please notify the Casal2 Development Team of any issues with or errors in Casal2. Please contact the `Casal2 Development Team`. See Section 18.2 for the guidelines for reporting issues. Note that Casal2 is a complex program, with many different options and possibilities, and we may not be able to provide any useful help if you submit an error report that does not follow the guidelines.

1.9 Technical details

The source code for Casal2 is available in the GitHub repository at `https://github.com/alistairdunn1/CASAL2`. Casal2 is compiled on GitHub using GitHub Actions on host operating systems Microsoft Windows Server 2019 and Ubuntu 20.04.

Casal2 was compiled on Linux using `gcc` (`https://gcc.gnu.org`), the C/C++ compiler developed by the GNU Project. The 64-bit Linux version was compiled using `gcc` version 10.3.0).

The Microsoft Windows (`https://www.microsoft.com`) version was compiled using TDM-gcc (`https://jmeubank.github.io/tdm-gcc/`) using `gcc` 10.3.0 (`http://gcc.gnu.org`). The Microsoft Windows (`https://www.microsoft.com`) installer was previously built using the Inno Setup (`https://jrsoftware.org/isdl.php`). **Note:** for some previous gcc versions, there have been issues related to threading. This was indicated by failed unit tests which rely on threading such MCMCHamiltonian etc.

Casal2 includes several minimisers; different minimisers may perform better for some models than others. These include both numerical differences minimisers and automatic differentiation minimisers. Numerical differences minimisers will usually work for most problems, albeit more slowly than auto-differentiation based minimisers. The numerical differences minimisers are:

1. Numerical differences: A minimiser that is closely based on the main algorithm of Dennis Jr and Schnabel (1996), that uses finite difference gradients.
2. deltadiff: A multithreaded version of the numerical differences, but with *tan* rescaling instead of *arcsin* and available for use with the Hamiltonian Monte Carlo MCMC algorithm.
3. DESolver: The differential evolution solver (Storn and Price, 1995), based on code by Lester E. Godwin of PushCorp, Inc.

There are two auto-differentiation minimisers, both based on ADOL-C. These are

1. Betadiff: A minimiser using an older version of ADOL-C (v1.8.4) that was used as the automatic differentiation minimiser in CASAL (Bull et al., 2012), with the same minimising algorithm as used for the finite differences minimiser above, but based on the gradient from the auto-differentiation chain.
2. ADOL-C: A minimiser using version of ADOL-C (v2.5.1) (Wächter et al., 1996);, with a similar minimising algorithm as used for the finite differences minimiser above, but based on the gradient from the auto-differentiation chain.

The random number generator used in Casal2 uses an implementation of the Mersenne twister random number generator (Matsumoto and Nishimura, 1998). This functionality, the command line functionality, matrix operations, and a number of other functions use the Boost C++ library

(Version 1.71.0).

Note that the output from Casal2 may differ slightly on the different operating systems and operating system versions due to different precision arithmetic or other platform-dependent (including CPU hardware) implementation details. In particular, the implementation of the standard C++ library `math.h` differs slightly on different platforms, and hence the results from different platforms may be different.

Casal2 uses unit tests and post-compile model validations to verify the source code. Unit tests of the underlying Casal2 code are run at build time using the Google Test and Mock unit testing and mocking 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. Post-compile models are run using Python scripts at the end of every build and with each commit to the GitHub repository. See Appendix B for more information.

2 Model overview

Casal2 is a generalised age- or length-structured population dynamics modelling framework for undertaking age- or length-structured integrated assessments (Maunder, 2013). Casal2 allows for multiple sources of information to be combined into a single analysis using a statistical framework so that error sources are fully propagated into the uncertainty in the outcomes. The model follows cohorts as numbers-at-age (or numbers-at-length for length-structure models) through time, recording the changes that occur from the population dynamic processes.

Casal2 is run from the console window on Microsoft Windows or from a terminal window on Linux. Casal2 has two sources of information: the *input configuration file* which defines the model structure, provides observations, defines active parameters, and specifies outputs (reports); and the run-time mode (e.g., estimate active parameters, run projections, etc..) that is given by command line options and arguments (see Section 3 for specific details). Typically a 'run' will simply run the model (without estimation) and generate the expected values and other models outputs from the parameter values assumed, and an estimation will minimise the model objective function to derive the 'best fit' parameters.

A Casal2 model is defined by its initial conditions and the processes that occur within and to the population over the years of the model run. The annual cycle defines the time steps that occur within each year and the order of processes within each time step. At each point in time, the model updates the *state* of the model, where the state consists of two parts, the *partition*, and any *derived quantities* requested.

The *partition* is a representation of the population at each time step, and can be considered a matrix of the numbers of individuals within each category (i.e., row) and at each age (i.e., column). The partition will change after each process, and hence after each *time-step* of every year. The rows of the partition (categories) and columns (lengths) define the population structure. For example, categories can define males and females, area, and/or maturity stage. Note that any user-defined category or category label is possible, for example species, stock, tagged status. The number of categories, what they represent, and how they interact is completely defined by the user. The model records the numbers of individuals within each category and length (e.g., for model with a two sexes, the numbers of males and females at length). In general, cohorts are added via a recruitment event, are aged annually, and are removed from the population via various forms of mortality (e.g., fishing or natural mortality).

A *derived quantity* is a value that is calculated for the partition at some point in time. An example of a derived quantity is spawning stock biomass (*SSB*) – the sum of the biomass of individuals that are mature (or spawning) at some specific point in the annual cycle. Unlike the partition, which is updated in each time step, a derived quantity calculates a single value for each year of the model run. Hence, derived quantities are a vector of values over the time period represented by the model. Each derived quantity can be reported or used as an input into a process. The most commonly used derived quantity is spawning stock biomass (*SSB*) in the stock-recruitment relationship which determines recruitment of the population into the model. Another example might be a density dependent mortality process for a species that is based on the biomass of another species in the model.

Observations are the data that was observed for some aspect of the population. Each observation block includes the observed values, the sampling distribution, relationship with the partition, and the time within the model that these occur (time step and year). For example, indices of abundance or biomass from a research survey, or length compositions from a commercial catch in a fishery. The partition is queried to generate the expected values for the observations, and then the sampling distribution and sample sizes are used to calculate their likelihood. In broad terms, the model

parameters are estimated to provide the best fit of the expected values to the observations, by minimising an objective function. Best fit is judged by the lowest objective function value, with the objective function equal to the sum of the negative log-likelihood values, the priors, and any model constraints (penalties). The evaluation of observations and the calculation of the expected values for each observation type is described in Section 7.

The method that Casal2 uses to find a minimum, the parameters to estimate and their priors is given in the estimation section, Section 6. This includes the choice of minimiser, MCMC algorithms and associated parameters, as well as any transformations and penalties used to constrain the model.

Outputs (reports) are defined in the report section. Casal2 has a large number of reports to summarise or generate a variety of output for a given model. See Section 8 for more information.

The model, its population structure, observations, methods of estimation, and output reports are all defined in the input configuration fileṪhe run mode of Casal2 is determined by the command line arguments given when 'running' Casal2.

The input configuration file is a text file with the Casal2 commands and subcommands. The input configuration file completely describes a model implemented in Casal2. See Sections 9, 10, 11, and 12 for details of Casal2's command and subcommand syntax. The default name for this file is `config.csl2`, however any file name can be used if given as an argument to the command line when calling Casal2. Generally, it can be useful to split the input configuration file into a number of smaller files using the `@include` command. We recommend using separate files for the different section for the configuration commands and subcommands to assist readability.

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 input file that is supplied to Casal2.

The input configuration file is required and defines the model structure, processes, observations, parameters (both the fixed parameters and the parameters to be estimated), and the requested reports (outputs).

By convention, the name of the input configuration file ends with the suffix `.csl2`. However, any suffix is acceptable. The default name for the input configuration file is *config.csl2* and if used it does not have to be specified as one of the command line arguments to to Casal2. Note that the input configuration file can include other files so the specification can be split into parts, e.g., files for specifying information on the 'population', 'estimation', 'observation', and 'reports'.

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.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 3.3.

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 3.3 for the list of possible arguments. Casal2 will print output to the screen.

For both 64-bit Linux and Microsoft Windows, we recommend using the zip files available on `https://github.com/alistairdunn1/CASAL2` for Microsoft Windows and Linux; there is also file 'Casal2.tar.gz' for older versions of Linux. Running Casal2 on a system requires the main binary (casal2.exe on Windows, or casal2 on Linux) and the associated dynamically linked libraries (DLL) for Windows or shared objects (.so) for Linux to be installed into appropriate directories, and cannot be (easily) run by copying the binary to a working directory. Casal2 is not available for 32-bit operating systems or MacOS.

3.2 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 a file or separate 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 standard output. The header consists of the program name and version, the arguments passed to Casal2 from the command line, the date and time that the program was called (derived from the system time), the user name, and the machine name (including the operating system and the process identification number). This information can be used to track output across runs, dates, and versions of Casal2.

3.3 Command line arguments

Casal2 is called using:

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

where

`-c config_file` Define the input configuration filefor 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 and version details (including the location of the GitHub repository from which the source code was compiled)

`-r [--run]` *Run* the model once using the parameter values in the input configuration file, or optionally with the free parameter values from the file specified with argument `-i` or `-I`.

`-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 free parameter values from the file specified with argument `-i` or `-I`.

`-E [--Estimate]` ***filename*** Do a point *estimate* and generate an MPD file (i.e., a file containing the free parameters and the covariance matrix). As with `-e`, this uses the values in the input configuration file as the starting point for the parameters to be estimated, or optionally the free parameter values from the file specified with argument `-i` or `-I`.

`-p[-profiling]` Do an objective function *profile* using the parameter values in the input configuration file as the starting point, or optionally with the free parameter values from the file specified with argument `-i` or `-I`

`-m [--mcmc]` Do an *MCMC*. An estimate run is fist carried out to estimate the covariance matrix for the MCMC proposal distribution, using the values in the input configuration file as the starting point for the parameters to be estimated. Optionally the free parameter values from the file specified with argument `-i` or `-I` can be used as the starting point.

`-M [--mcmc-from-estimate]` ***filename*** Do an *MCMC* run using the covariance and free parameters in the MPD file.

-R [--resume] ***filename*** Resume a previously stopped *MCMC* run using the covariance and free parameters in the MPD file. Additional arguments must be supplied to specify the sample and objective files from the previous MCMC with `--objective-file` and `--sample-file`.

-f [--projection] *n*. Project the model *forward* in time using the parameter values from a file specified with the argument `-i` or `-I`. Projections are repeated for each parameter set (i.e., for each line of data in the free parameter file) *n* times. Typically, the MCMC sample output will be used with `-i`.

-s [--simulation] ***n*** *Simulate n* sets of observation data 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` or `-I`.

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 8 for details about the format of *`filename`*).

-I [--input-force] ***filename*** *Input* one or more sets of parameter values from *`filename`*. This contains both the free parameters and also force the *overwrite* addressable (non-estimated) values in the input configuration file (see Section 8 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 8 for details about the format of *`filename`*)

-O [--Output] ***filename*** *Output* and append a report of the free (estimated) parameter values in a format suitable for `-i` *`filename`* (see Section 8 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)

-L [--loglevel] ***arg*** Set the level for information or logging messages from Casal2. Valid options are (from more verbose to less verbose) trace, finest, fine, medium, info, important, and warning. The default is 'info' (see Section 18.1 for more information).

-t [--tabular] Print `@report` in tabular format (see Section 8 for more information).

--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, printing the inputs for this process (should be consistent with syntax section).

-V [--verifylevel] ***arg*** If Casal2 exits with a verify message (the default), then it will halt. If *arg* = *warning* Casal2 will complete the model run and print the verify statement.

Combinations of these command line arguments can also be implemented. Examples of some useful ones are below.

`casal -r -i par.file > multi_run.out` will conduct multiple model runs, one for each row of parameters in `par.file`. This can be useful for investigating the effect of individual parameters in the model or summarising profiled outputs.

`casal -e -i par.file > multi_estimate.out` will conduct multiple estimation routines. One for each row of parameters in `par.file`. This can be useful for assessing convergence to a global minimum. All base models should be run from multiple starting parameter values to assess model convergence/sensitivity to starting values.

`casal -s 10 -i par.file > multi_simulation.out` This command instructs Casal2 to simulate 10 sets of simulated data sets for each each row of parameters in `par.file`. The `-s` component adds observation error in simulated data sets through the likelihood distribution assumptions and the `-i` adds parameter uncertainty into the simulated data sets if each row differs.

`casal -r --loglevel trace > run.log 2> run.err` This command runs the Casal2 model with parameters based on the configuration files and will print logging information into the file `run.err` (useful when debugging models). See Section 18.1 for details on logging.

3.4 Constructing the Casal2 input configuration files

- the description of the population structure, dynamics, and parameters. See Section 5,
- the estimation methods and estimated variables. See Section 6,
- the observations and their associated properties and likelihoods. See Section 7, and
- the reports that Casal2 will output. See Section 8.

Note that input configuration file files can *include* other input configuration files to assist in file management, using the command `!include "filename"`. See Section 13 for more details.

3.4.1 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 only 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` or `@categories`)

Apart from `!include`, commands start with an `@` in the first column (i.e., may not have a space or tab character before them on the line). After each command, the subcommands are listed and must occur before the next command. Otherwise, the commands and subcommands are free form with each command or subcommand on a separate line (see Section 3.4.3).

Commands that have a label must have a unique label, i.e., the label cannot be used for more than one command. Casal2 checks and will report an error if two commands of the same type have the same label. The labels can contain alpha numeric characters, period ('.'), underscore ('_') and dash ('-'), but cannot start with a double underscore ('__'). Labels that start with a double underscore are reserved, and used for internal reports that Casal2 can automatically generate in some circumstances. Otherwise labels must not contain white space (tabs or spaces) or any characters that are not letters, numbers, dashes, periods, or underscores. For example,

```
@process NaturalMortality
```

or

```
!include MyModelSpecification.csl2
```

3.4.2 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 *types* for each subcommand are defined in Section 14.1.3, 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 unexpected 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 white space (tabs or spaces). Arguments of type integer range must contain a colon (`:`) and no white space (tabs or spaces).

Parameters are defined in the population section and most (but not all) numeric parameters can be estimated. See Section 14.1.3 for the list of available parameters and if they are can be estimated. Note that parameters will only be estimated if requested using an `@estimate` command, and are otherwise treated as a constant.

Parameters can also be addressable, i.e., they can be referred to within another command or command block by using their addressable name. See Section 14.1.3 to determine of a subcommand is addressable.

3.4.3 The command block format

The command block is a basic unit within the input configuration file. Each command begins with the symbol `@` and then the command name, usually followed by a user defined label or a valid argument. The end of each command block is denoted by the start of the next command block or end of the file. For example, the layout of a input configuration file will be

`@command label`

`first_subcommand argument`

`second_subcommand argument`

... etc.

`@another_command label`

`another_subcommand argument`

`another_subcommand argument`

... etc.

Note that subcommands can be in any order within each command block. And command blocks can be in any order within the input files, except `@model` — this must be the first command block encountered by Casal2.

Blank lines are ignored, as is extra white space (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 white space. Each input file must end with a carriage return.

Commands, subcommands, and arguments in the input configuration files are not case sensitive. However, labels and variable values are case sensitive. Note that on Linux (unlike Microsoft Windows) specification of any file names or file paths will be case sensitive.

3.4.4 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
*/
```

3.4.5 How to reference parameters

All parameters have a unique name, allowing it to be referenced in other command blocks. When Casal2 processes the input configuration file it translates each command block (see section 3.4.3) and each subcommand block into an object, each with a unique parameter name. For commands, this parameter name is simply the command label. For subcommands, the parameter name format is one of the following:

`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.

For example, the parameter name of a process of instantaneous mortality (i.e., natural mortality) is the subcommand `m` of a `@process` of type `mortality_constant_rate`, i.e., the command block may be

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

`process[NaturalMortality].m` is the unique reference for the vector of male then female natural mortality values (**note:** order will follow categories order). To reference just the 'female' value then the form is `process[NaturalMortality].m{female}`.

3.5 Reading a command block

Here, we illustrate reading a command block using two important commands, `@process` and `@estimate`.

The command `@process` specifies a process that can be used in the model. There are a fixed set of predefined processes (subroutines in C++ code). The way to specify which process is used is with the `type` subcommand. Processes can take one or more parameters and some will need other data to be supplied as well. Some parameters are mandatory and others can take a default value if they are not specified.

For this example we have categories male and female, and two fisheries, line and pot. The command block starts with a*@process*:

```
@process Fishing
type mortality_instantaneous
```

This sets up a process block using the *mortality_instantaneous* process which simultaneously depletes the population by natural mortality and from two types of fishing. Its label is *Fishing*.

Next we specify the values for natural mortality (*m*), an argument for this process, to 0.17 and specify that fisheries acts on all categories. Note there are two values for natural mortality, one for each category. The parameters *m* can be estimated, if required. The command block fragment:

```
m 0.17 0.17   # natural mortality  for each category
relative_m_by_length One One # natural mortality multiplier
categories *  # fishing acts on all categories ("*" shorthand for male female)
```

Catches are supplied via a *table* format using three columns: one for year and one for each of the two fisheries, which take the labels *line* and *pot*. Column names are on the first line of the table and these columns can be in any order,

```
#catches
table catch         # define catches by fishery in table format
year line pot       #names columns so can identity catch for each fishery
2000 1000 2000      # catches by year
2001 500  1000
2002 1000 5000
end_table           # end of table marker
```

Other information required is supplied in the methods table which has a fixed number of columns (again these can be in any order), one for each piece of information needed to specify a fishery. The method column defines the fishery name which is used in the catch table and also in other observations like length composition from that fishery. The categories that the fishery operates on (all in this case, but it could be just males for one and females for the other) are in the category column, the fishing selectivity to be used is given as a selectivity block name which is define somewhere else in the files, U_max is the maximum exploitation rate that is allowed in any year, then the time step the fishing operates in, and lastly the block name of a penalty function that is used to penalise estimable parameter values that result in the supplied catch not being caught. Again, the penalty block is define elsewhere in the files. After the row with the column names, there is one row for each fishery:

```
table method        # supply arguments and name selectivity etc
method     category  selectivity    u_max    time_step      penalty
pot        *         potFSel        0.7      1              CatchMustBeTaken1
line       *         lineFSel       0.7      1              CatchMustBeTaken1
end_table
```

To estimate natural mortality, you need to supply an *@estimate* block with a reference name back to *m* in the *Fishing* block. For *@estimate*, *type* specifies the prior to be used in the estimation, which in this case is a normal distribution:

```
@estimate estimate.m
type normal # prior type
parameter process[fishing].m
# this is a comment
/*
Fishing is unique amongst the @process command blocks
so this defines the unique reference to the parameter m
*/
mu  0.2 0.2   #argument to prior = mean
sd  0.02 0.02 #another argument to the prior = standard deviation
```

Note that there are two *m* values, one for each category, so there are two priors specified. The *estimate* label *estimate.m* is often redundant, but it may be needed in some circumstances.

To estimate a common *m* over both sexes, we estimate one *m*, say the female category, and use the *same* subcommand to apply the same value to the male category *m*,

```
@estimate estimate.m
type normal
parameter process[fishing].m{male}
# {} is used to index one or more elements in a vector
same process[fishing].m{female} # set female value = male estimated value
# The mean of the prior
mu  0.2
# The standard deviation of the prior
sd  0.02
```

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`). Note this is still an experimental feature.

3.7 Logging and verifying Models

Casal2 has a number of standard information, warning, and error message outputs. Additional logging and debugging information is available using the `--loglevel` or `-L` command line option. See Section 18.1 for details on logging.

Casal2 also applies sanity checks on model configurations. These can be bypassed using the command line option `--verifylevel` or `-V`. These sanity checks are based on expected model structures, i.e., in age-based models Casal2 verifies models have an ageing process. Currently only a few sanity checks are implemented.

3.8 Validating models between different versions of Casal2

Casal2 can validate or check addressables parameters for testing and validation using the `@assert` command. Asserts check the value of a specific addressable (for example, an observation, parameter, or the objective function) against a user defined predefined value. See Section 14 for more information.

Asserts are one of a number of the internal and system tests used by Casal2 to ensure consistency across versions and revisions. Casal2 also uses unit tests and post-compile model validations to verify the source code. See Appendix B for more information.

3.9 Casal2 exit status values

When Casal2 is run, it will either complete its task successfully or output an error. 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,

```
[ERROR] At line 15 in Reports.csl2: Parameter '{' is not supported
```

4 Partition & Categories

Dividing the population into categories is fundamental to modelling the dynamics of a fish stock. The grouping of the population into categories and either ages or lengths is called the partition.

In Casal2 the concept of user defined categories allows for flexibility in grouping of categories or parts of the modelled population. Note that Casal2 does not know about sex or area and their properties; these are explicitly defined by the user by specifying processes that act on the categories. The cost is that users need to follow good practice to achieve clarity and readability of the input files, i.e., poor specifications can result in input files that are more difficult to understand.

CASAL had a fixed set of hard-wired categories (e.g., factors like sex, maturity, area, or stock) and each category type had a predefined set of allowed processes (or transitions in CASAL-speak), e.g., immature fish moving into the mature category (Bull et al., 2012). This made sense when CASAL was coded, but now it is seen as a limitation, e.g., changing sex was not allowed and only male and female sexes were allowed, not an unknown sex that sometimes occurs in data.

4.1 Specifying the partition using categories

A key element of the Casal2 model is the partition which holds the current state of the population. The partition can be conceptualised as a matrix, where each row represents a category and the columns are the length classes (Figure 4.1). Each row represents all individuals in that category as a numbers-at-length vector. There must be at least one category defined for each model.

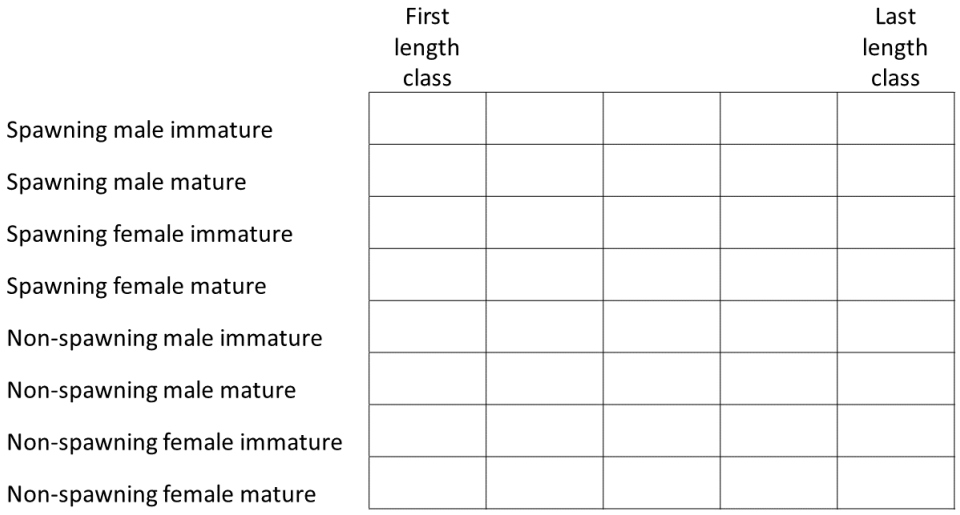

Figure 4.1: A visual representation of the partition for a simple length-based model.

The categories can include combinations of levels from one or more factors such as sex, maturity state, area, stock, or even species. Casal2 has no predefined categories; *all* categories are defined by the user. Note that the partition only has the current state of the model; past states are not kept (*See* the section on derived quantities about saving summary values from the partition, p. 38).

To illustrate categories, consider a model of a fish population with two fisheries, one on spawning fish at the spawning grounds and another on the non-spawning population in the rest of the stock area. The mature fish will 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 by factors sex (levels *male* and *female*), maturity (levels *immature* and *mature*), and area (levels *spawning* and *non-spawning*). So the partition has 8 rows of numbers-at-length, for 2 sexes $\times$ 2 maturity levels $\times$ 2 areas.

These categories are specified in a categories block which starts with a @*categories* line followed on the next

line by a *format* subcommand that specifies the factors to use and their order. Factor names are user defined and have no intrinsic meaning in Casal2.

The command block for this example is:

```
@categories
format area.sex.mature
names spawn.male.immature spawn.male.mature spawn.female.immature spawn.female.mature
    nonspawn.male.immature nonspawn.male.mature nonspawn.female.immature #all on one line
    nonspawn.female.mature
```

Note the "." syntax to separate the factor names.

Next comes the *names* subcommand which specifies the combinations of levels that makes up each category. In a sense, the *format* subcommand is not needed since the *names* subcommand can define all categories. However, *format* allows a more digestible and shorter syntax to define categories here and in other blocks such as matching observation to categories that provided the data (including combinations of categories, e.g., length compositions that combine both sexes).

The *names* subcommand can also be specified with:

```
names spawn,nonspawn.male,female.immature,mature
```

which defines the categories above in a more efficient manner, (again, note the "." to separate the factors and "," to separate the levels within each factor (see the next section for more details). A visualisation of the partition is in Figure 4.1.

When using this short-cut syntax in *names*, the order of level combinations is for the levels of the first factor to change the slowest, then the next factor will change faster, and so on with the last factor to changing levels the fastest. The order is important because linking categories to their characteristics, e.g., growth curve or selectivity, is done in other subcommands where these must be specified in the same order.

To exclude unused categories from the partition, the long form must be used in the *names* subcommand, e.g., to exclude *spawn.female.immature* and *spawn.male.immature* since they are never in the spawning area.

To make recruitment to enter the partition in the non-spawning area, use

```
@categories
format area.sex.mature
names spawn.male.mature spawn.female.mature nonspawn.male.immature nonspawn.male.mature
    nonspawn.female.immature nonspawn.female.mature
```

4.2 Shorthand syntax for categories

Some specifications have long lists of categories or years or initial values for parameters and the like, e.g., for YCS from 1900 to 2019, 120 years and 120 initial vales of YCS must be specified; this is hard to do by hand and it can be error prone as well as difficult to match values for each year. Here, the range short cut (*:)* can be used so the the year specification is *1900:2019*, and the multiplier short cut (**)* to give the initial values specification as *1*120*.

There is also shorthand notation for categories since each category can be quite complicated. First use the `format` subcommand in the *@categories* block to define the factors that make up the sections of the category names. A "." (period) character delineates each factor and this structure allows a shorthand syntax to compose category names.

The `names` subcommand is used to list the category names. Sections within the shorthand syntax for *names* are required to match the order of factors in the *format* subcommand so Casal2 can organise and search on them. In these sections, levels for each factor use the "list specifier" and range characters, e.g.,

```
@categories
format sex.stage.tag # 2 sexes, 2 stages, tag years 2001 to 2005 = 20 categories

names male.immature # Invalid: No tag information
```

```
names female # Invalid: no stage of tag information
names female.immature.notag.1 # Invalid: Additional format segment not defined

names male,female.immature,mature.notag,2001:2005 # Valid shortcut

# Without the shorthand syntax these categories would be written:

names male.immature.notag male.immature.2001 male.immature.2002 male.immature.2003 male.
    immature.2004 male.immature.2005 male.mature.notag male.mature.2001 male.mature.2002 male.
    mature.2003 male.mature.2004 male.mature.2005 female.immature.notag female.immature.2001
    female.immature.2002 female.immature.2003 female.immature.2004 female.immature.2005 female
    .mature.notag female.mature.2001 female.mature.2002 female.mature.2003 female.mature.2004
    female.mature.2005
```

The shorthand syntax available are:

- * Specify all categories
- + Categories join, e.g., *categories* *+ joins all categories together into one unit; *categories male+female* specifies that the observation covers both sexes combined.
- *:* Specify a range of integers [int1]:[int2], e.g., *2000:2005* expands to *2000 2001 2002 2003 2004 2005*
- Lists using "," [item1],[item2],[item3], e.g., *male,female,unsexed* are the levels for the factor *sex*.
- Repeats a number or label: [number | label] * [integer], e.g., *1 * 5* $\rightarrow$ *1 1 1 1 1*
- *format*=[*X*]=[*x*]=[*int*] [*factor*]=[*level*]=[*year range*], e.g., *tag=2001=1999:2003* the categories with level 2001 in the tag factor are accessible from year 1999 to 2003 inclusive.
- *[]* replace label to a command block with the block defined inline, e.g., *catchability [q = 1e-5]* rather than *catchability CHATq* where *CHATq* labels a command block somewhere in the input files

Example of specifying categories using the short cuts:

This syntax is the long way:

```
@categories
format sex.stage
names male.immature male.mature female.immature female.mature
```

A shorter way to specify the exact same partition structure using *lists*:

```
@categories
format sex.stage
names male,female.immature,mature
```

Casal2 requires categories in processes and observations so that the correct model dynamics can be applied to the correct categories of the partition.

To combine/aggregate categories together, use the "+" special character. For example, this feature can be used to specify that the total biomass of the population is made up of both males and females.

For example,

```
@observation CPUE
type biomass   # observation using an index of biomass
categories male+female
        ...   # other subcommands to link index to the fishery etc
```

This combination/aggregation functionality can be used to compare an observation to the total combined population:

```
@observation CPUE
type biomass
categories *+
 ...   # other subcommands to link index to the fishery etc
```

If the levels `male` and `female` are the only categories in a population (i.e., factor *sex*), then this is the same syntax as the command block above it.

Shorthand syntax can be useful when applying processes to a specific group of categories from the partition.

For example, to apply a spawning migration to the mature categories in the partition given the partition definition

```
@categories
format area.maturity.tag
names north,south.immature,mature.notag,2001:2005
```

To migrate a portion of the mature population from the southern area to the northern area:

```
@process spawn_migration
type transition_category   # process to move fish from one category to another
from format=south.mature.* # move all south mature fish, both notag and tagged fish
to format=north.mature.*   # into the relevant north categories
```

An easy way to determine if you have specified the syntax correctly is to look at a report. Casal2 will expand most shorthand category labels in reports, and this can be used to check the order that Casal2 has assumed, and that these have been specified in the correct order for other related parameters .

4.3 Referencing vector and map parameters

To build relationships between command blocks, Casal2 uses a referencing system so that blocks and parameters within blocks can be accessed. In its simplest form, command blocks are referenced by their label. To access specified parameters within a command block, the syntax used is:

```
<syntactic element>     #<>  enclosing a description of the element

# most used version
<block type>[<label of block>].<parameter name>
# e.g., identify a fishery
<block type>[<label of block>].method_<parameter name>

## Examples
# recruitment multiplier (ycs) parameter in the process block called recruitment
process[recruitment].recruitment_multiplier

# natural mortality in the process called Fishing
process[Fishing].m

# pot fishery in the process called Fishing
# it is usual to define all fisheries in one
# mortality process block so we need a way to
# identify each one
process[Fishing].method_pot
```

Parameters can be scalars (one value), vectors (several values), or maps. A map consists of two vectors: one containing a key value (for searching or uniquely indexing), and another vector that contains values

associated with the index, e.g., for specifying recruitment multiplier values for each year, the years are the key (or index). To reference one or more components of a vector or map use the {} syntax. This may be needed when specifying which element(s) in a vector or map are to be estimated.

An example of a map parameter is `recruitment_multipliers` in a recruitment process

```
@process   WestRecruitment
# Beverton-Holt function
type       recruitment_beverton_holt
# initial values of the recruitment_multipliers (YCS) (a vector with 9 values)
recruitment_multipliers 1 1 1 1 1 1 1 1
standardise_years 1975:1983

# An alternative method to specify a sequence of values
# use an asterisks to represent a vector of repeating integers
recruitment_multipliers 1*8
```

To specify that only the last four years of the recruitment multipliers (YCS) parameter `process[WestRecruitment].recruitment_multipliers` are to be estimated:

```
@estimate RecMult    # RecMult is a label to identify this block
# estimate 4 values only: 1980, 1981, 1982, & 1983
parameter process[WestRecruitment].recruitment_multipliers{1980:1983}
```

To estimate a common value for a block of years in a map parameter use the *same* subcommand. We illustrate the idea within the process `@time_varying[label].type=constant`, where we want to fix q over a specified block of years, 1992 to 1995.

First specify the relationships in a `@time_varying` block:

```
@time_varying q_step1
# specify a set value for a year
type          constant
# parameter ref for q in block Fishq
parameter     catchability[Fishq].q
# or 1992:1995 = key into value
years  1992 1993 1994 1995
value  0.2 0.2 0.2 0.2
# or 0.2*4, initial values of q
```

Next, to estimate only one q value for the time block, pick one element of the map (say 1992), and then force all other years to have the same value:

```
@estimate q_block_1992
# estimate this one
parameter time_varying[q_step1].value{1992}
# set these to the value for 1992
same      time_varying[q_step1].value{1993:1995}
# uniform prior on q
type      uniform
lower_bound 0.1
upper_bound  10
```

Keys are restricted in Casal2 to years and categories. An example using categories as a key in a map:

```
@category
factor sex
names male female

@process recruit
categories male female
# natural mortality values indexed by categories
m  0.17 0.17
...

@estimate M
# prior = uniform
type uniform
# estimate male M, "male" is a level for factor sex
parameter recruitment.[m]{male}
# set female M to the same value as male's
same recruitment.[m]{female}
```

For vector parameters (i.e., no key values), the index is an integer starting with 1 for the first value, i.e., similar to R syntax. An example is the selectivity *all.values.bounded* which can be defined by:

```
@selectivity MatSel
type         all_values_bounded
# lower bound at age (if age-based) or length class (if length-based) of 2
L            2
# upper bound at age (if age-based) or length class (if length-based) of 4
H            4
# 3 values, one for each 2, 3, and 4
v            0.1 0.2 0.7

@estimate  mature
# prior = uniform
type       uniform
# estimate the 2nd value only, i.e., at 3
parameter selectivity[MatSel].v{2}
# lower parameter range
lower_bound 0.1
# upper parameter range
upper_bound 1.0
```

The integer *2* cannot be used to specify the *q* parameter for 1993 in the above example labelled *q_block_1992*. This will pass the syntax test, but it will fail at the validate stage in Casal2.

In-line declaration, avoiding extra command blocks In-line declarations can help shorten models by defining `@` blocks within the subcommand line instead of having a label that points to a command block define somewhere else in the input files.

For example, catchability for a CPUE index can be defined in-line:

```
@observation chatCPUE
type biomass                    # biomass index
catchability [q=6.52606e-005]  # define catchability here
categories male+female         # index cover both sexes together


@estimate chatCPUE_q
```

```
parameter catchability[chatTANbiomass.one].q # how to reference q
type uniform_log     # prior
lower_bound 1e-2
upper_bound 1
```

In the above code catchability is defined and estimated without explicitly creating a `@catchability` block.

5 The population section: model structure and the population dynamics

The command and subcommand syntax for the population section is given in Section 9.

5.1 Introduction

This section shows how to specify a model for the population dynamics. It describes the model time and length scope, the population processes used (e.g., recruitment, ageing or growth transition, migration, and mortality), the selectivity ogives, and how to set values for their associated parameters, or starting values if they are going to be estimated.

The basic structure of the population is defined in terms of its partitions and the succession of processes that act on them throughout a year. Casal2 assumes an annual cycle, i.e., rates like natural mortality are assumed to be for a year. To place certain processes or observations (e.g., a research survey) into the right part of the year, the year can be divided into one or more time steps, and each time step needs at least one process. Each time step can represent a specific period of the calendar year, or it can be an abstract sequence of events. Certain processes like natural mortality and growth can have a proportion of the effects of the process assigned to different time steps to crudely mimic seasonal effects, or fisheries that occur in short periods of the year, as well as place a survey within the year relative to the proportion of annual natural mortality that has occurred (see Section 5.4).

The *state* is the current status of the population at any given time and it can change one or more times during the 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 partition is key to the state, but it has no ”memory”. Thus, other information must also be kept, such as the mature biomass from a previous year or time step to calculate the recruit numbers into the first length class via the spawner-recruitment relationship. Quantities like mature biomass are defined as *derived variables* and are calculated for each year of the model. However, the *derived variables* record only summary information from the partition at a specified time step and year.

Processes can change the partition and, for example, include recruitment, natural mortality, fishing mortality, ageing (in age-based models) or a growth transition (in length-based models), migration, and maturation. These processes are repeated for each year of the model.

The specification 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 impacts 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 4.2.

Topics covered are:

- The model scope, such as the length covered, the years over which the model runs, and the end year for projections (Section 5.2);
- Linking processes to each category;
- The number of time steps and the processes that are applied in each time step (Section 5.2.1);
- The specification of and the parameters for the population processes: processes that add or remove individuals from a partition, or shift individuals between length classes and categories in a partition;
- The initialisation process: the state of the partition at the start of the first year;
- Defining selectivity ogives and linking them to observations;
- 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

5.2 Model scope and structure

The model needs scoping for length classes and year covered. This is done in the `@model` command block.

Each Casal2 model requires:

- The length classes used
- Whether the maximum length class is a plus group
- The mean length of the last plus group if its a plus group
- The start and final year
- The names of all of the categories

The length classes used using the length bins defined. The last group can be a plus group and, if so, the mean length of this group must be defined. The model is run from the start year through to the final year. It can also be run past the final year to project the state of the population through the final projection year.

An example of how to specify a potential model with two categories is outlined below; the `@model` and `@categories` blocks are:

```
@model
start_year 1981
final_year 2000
projection_final_year 2010
base_weight_units     tonnes
length_bins     1:68
length_plus True
length_plus_group 69
initialisation_phases Equilibrium_phase
time_steps            step1 step2 step3

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

This model runs for 20 years, starting in 1981, and will do a projection over 10 years for a population with length classes 1–68, with the last class being a plus-group with mean length 69 cms. Each year is divided into three time-steps. The categories are male and female (i.e., there is one category factor, labelled *sex*) and each category has an associated growth transition matrix .

Whist Casal2 generally uses generic formulation, it does have some specific population concepts, in this case, the growth transition matrix can be different for each category. These are specified in the growth increment blocks that define how individuals size increments with each growth episode, which in this example are command blocks starting with `@growth_increment male_growth` and `@growth_increment female_growth` that are placed elsewhere in the input files (not shown).

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 5.2).

Derived quantities are an 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`, 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. See Section 5.4 for more details.

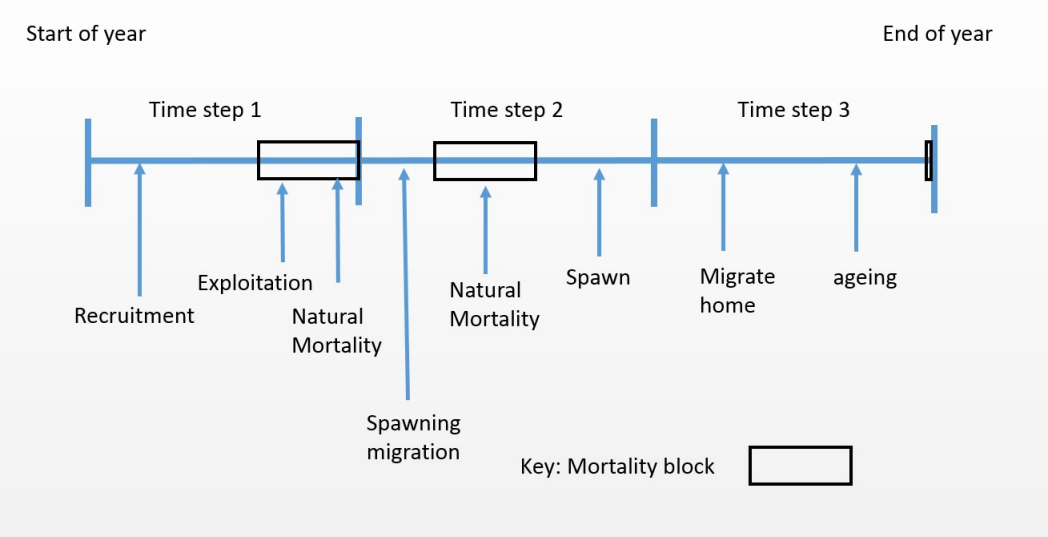

Figure 5.1: A example sequence for an annual cycle for an age-based model.

5.2.1 The implicit annual cycle

There is an implicit annual cycle that orders the sequence of processes within the year, but there is no command block as such. The implementation is by ordering processes within the time-steps. This sequence is repeated for every year. Time steps are used to break the year 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 5.3.2); 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 (5.2.2).

Figure 5.1 shows an example of the annual cycle for an age-based models using three time-steps (for length-based models, the example is the same, except that the growth process would be replaced by a growth increment process).

This would be specified using `@time_step` block:

```
@model
time_steps step1 step2 step3
```

This gives the order and labels for each time step, i.e., 3. Processes are sequenced using order within the `@time_step` block:

```
@time_step step1
processes Recruitment Fishing

@time_step step2
processes Spawn_migration Fishing

@time_step step3
processes Home_migration Growth
```

The *Recruitment*, *Fishing*, *Spawn_migration*, *Home_migration* and *Growth* are all labels of command blocks that defines a process (see Section 5.3 for the list of available processes). The order that the processes are executed is in the same order as specified. The process *Fishing* could be the process type `Instantaneous_Mortality` (Section 5.3.2) which takes natural mortality as a parameter as well as specifying

the catches in the time-steps, so it is possible to have all catch taken in time-step *step1* with some natural mortality, and no fishing in time-step *step2* where the rest of the natural mortality occurs.

Although the process *Spawn* represents a biological process, spawning, in the Casal2 model it is the time that the spawning stock biomass (*SSB*) is calculated since this is needed to calculate recruitment if there is a spawner-recruitment relationship. A related concept is maturity which can be in the partition, so there needs to be a process to transfer immature fish into the mature category, but it is only indirectly related to spawning. Hence, in modelling, spawning is not a process that affects the partition directly, but it the time to calculate the *SSB* which must be defined as a derived quantity (from the partition). Hence, *Spawn* is located in Figure 5.1.

To calculate the *SSB* a `@derived_quantity` command block is needed in which the "timing" of the *SSB* calculation in terms of which time-step and the proportion of natural mortality within it is specified (5.4).

5.2.2 The initialisation phases

Initialisation is the process of determining the model starting state at the start of the first year (`Start_year`. 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: run the model for a specified number of years to get the converged state.
- Derived: Use the analytical solution (i.e., faster than iterative) for the initial state, but it does not work with some processes (e.g., density-dependent migration)

Initialisation definitions start with specifying the initialisation label in the `@model` command block followed by a `@initialisation_phase` command block specifying the type and other settings:

```
@model
...     # other subcommands
initialisation_phase int_label

@initialisation_phase int_label
type iterative  #choose one from the list above
...             # specify option values
```

If needed, the processes used and their order in the initialisation are those specified in the annual cycle, but these can by changed by either excluding some processes or including others by using the `exclude_processes` or `insert_processes` subcommands in the *initialisation_phase* command blocks,

```
@initialisation_phase int_label
type iterative
exclude_processes Fishing
insert_processes step1(recruitment)=initialFishing
            # format: <step>(<insert before process label>)=<new block label>
...         # specify option values
```

where *Fishing* is the normal fishing process which defines natural mortality so when excluded, initialisation can use another value that incorporates some unrecorded fishing before the start of the assessment period by setting natural mortality to a higher value in the process *initialFishing*. The place to insert *initialFishing* is in the time-step labelled *step1* before the process *recruitment* which must be in that time-step (process label is enclosed in brackets). To insert at the end of the time-step use *()*, e.g. *step1()=initialFishing*.

For most models the most common type of initialisation phase to define an initial equilibrium age structure is `derived`, whereas for length-based models the only type available is `iterative`. Additional initialisation phases can be included by sequencing other phases one after another

```
@model
...     # other subcommands
initialisation_phase int_label int_label2


@initialisation_phase int_label
type derived    #choose one from the list above
...             # specify option values

@initialisation_phase int_label2
type iterative    #choose one from the list above
...             # specify option values
```

which may be faster overall since fewer iterations may be required used in the second phase. The order of applying each initialisation is that given in the `@model` command block.

The multi-phased initialisation allows for flexibility in the number and type of initialisation processes, for initialising a non-equilibrium starting state, or applying simple processes before applying more complex ones.

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 length class 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.

For multi-phase initialisation models, it is advised to include the `@report` of type `initialisation_partition`. This will print the partition at the end of each initialisation phase, which can be useful for assessing the impact of each phase on the partition.

```
@report initial_partitions
type initialisation_partition
```

Iterative Initialisation The `iterative` initialisation is a general solution for initialising the model, but can be slow to converge, depending on the model. Its value is that it can work on 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. In an age-based model, a period of about two times the maximum age is recommended to ensure convergence. In length-based models a longer time may be required depending on the nature of the growth increment process. Casal2 can report a convergence statistic to assist in determining if adequate convergence has been obtained.

In addition, the iterative initialisation phase can optionally be stopped early if the 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

$$\widehat{\lambda} = \frac{\sum_{i,j} \left| \text{element}(t)_{i,j} - \text{element}(t-1)_{i,j} \right|}{\sum_{i,j} \text{element}(t)_{i,j}} \tag{5.1}$$

where $\text{element}(t)_{i,j}$ denotes the numbers at time step t in category j and length class i.

Hence, for the 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                # do 50 annual cycle iterations
lambda 0.0001
convergence_years 20 40 # test for convergence at 20 and 40 iterations
```

A report on the outcome of the iterative convergence evaluation is available (`@report` of type `initialisation`). This will print the years when convergence was tested and the result of the convergence tests.

When a model is initialised with `derived` (for age-based models only) or `iterative` (for either age-based or length-based models), and recruitment is defined by B_0, the model initialises the partition with $R_0 = 1$. Once the initialisation phase is complete, it scales all the categories defined in each recruitment process by

$$N_{a,c} = N_{a,c} \times B_0^R / SSB^R \ .$$

where, R denotes each recruitment block and $N_{a,c}$ are categories defined in that recruitment block. For this case, it is advised to associate all categories to the recruitment so they are accounted for in this scaling process. If maturity is in the partition, it is not intuitive, but they must be defined in the recruitment dynamic with a proportion set = 0 (for more information on specifying this see Section 5.3.1). Casal2 will flag a warning if a model doesn't have all categories defined in the available recruitment blocks. A case where this can be ignored is in models with tagged categories, these categories don't exist during initialisation and so don't need to be scaled, and thus can be omitted from the recruitment definition. Casal2 will still output a warning for this, but can be ignored if users understand its purpose.

5.3 Population processes

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

Current population processes available include:

- recruitment (Section 5.3.1),
- growth (Section 5.5),
- mortality events (e.g., natural and fishing) (Section 5.3.2),
- category transition processes, i.e., processes that move individuals between categories while preserving their overall length structure (Section 5.3.3), and

There are two types of processes: (1) processes that occur across multiple time steps in the annual cycle, e.g., `mortality_constant_rate`, `mortality_instantaneous`, `growth`; and (2) processes that occur only within the time step in which they are specified.

5.3.1 Recruitment

The recruitment processes adds new individuals to the partition. Recruitment depends on the B_0 and R_0 parameters which are interpreted as the average spawning stock biomass and recruitment over the period of data when there is no fishing. The other factors needed are spawning stock biomass (*SSB*; see Section 5.4), stock-recruitment relationship and the CV for the prior on recruitment multipliers (the mean is mandated to be 1 over some specified year range). Thus, a *SSB* label may have to be included (pointing to a derived quantity).

In the recruitment processes, a number of individuals are added to a range of categories and length bins within the partition, with the overall number determined by the type of stock-recruitment process specified. If recruits are added to more than one category, then the proportion of recruits to be added to each category is specified by the `proportions` subcommand. 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. One view for fisheries is that recruitment usually refers to individuals "recruiting" to a fishery. This definition is used because there is generally not a lot of observations for younger fish between the time of spawning and being vulnerable to a survey or fishery for data collection.

The year offset for an age cohort between spawning and recruitment into the partition is required. The Casal2 parameter `ssb_offset` defines this offset year and is analogous to the CASAL parameter `y_enter`.

$$N_{y,l,j} \leftarrow N_{y,l,j} + p_j(R_y) * f_j(l,\mu,\sigma) \tag{5.2}$$

where $N_{y,l,j}$ is the numbers in year y and category j at length bin l, p_j is the proportion added to category j, $f(l,\mu,\sigma)$ is the density function of recruits among length bins for category j, and R_y is the total number of recruits in year y.

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

$$R_{y,j} = p_j(R_0) \tag{5.3}$$

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
length_bins 1
```

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 recruitment multipliers (YCS, also called year class strength), and the stock-recruit relationship $SR(SSB_y)$

$$R_y = (R_0 \times YCS_y \times SR(SSB_{spawn\_year})) \tag{5.4}$$

where

$$spawn\_year = y - \texttt{ssb\_offset} \tag{5.5}$$

Recruitment refers to an age cohort recruiting into the partition and may differ from the spawning year. See below on more information about `ssb_offset`.

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

$$SR(SSB_y) = \frac{SSB_y}{B_0} \Big/ \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB_y}{B_0}\right)\right) \tag{5.6}$$

The Beverton-Holt recruitment process requires a value for B_0 (or R_0) and SSB_y to calculate the number of recruits. A derived quantity (see Section 5.4) must be defined that provides the annual SSB_y for the recruitment process. B_0 is then defined as the value of the *SSB* calculated during initialisation. If a model has more than one initialisation phase, the user needs to supply the initialisation phase that calculates B_0. This is defined by the command `b0_initialisation_phase`. Casal2 will default to the last initialisation phase if users do not specify this command.

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

Recruitment during 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 either B_0 or R_0.

The length apportionment of recruits into the partition are derived using the normal cumulative function over the model length bins. This cumulative function is denoted by $f_j(l, \mu_c, \sigma_c)$. For each category this will calculate the probability of being in a length bin and will sum equal to one over all length bins. The mean and variance of this normal cumulative function can be category specific and is defined by the input parameters `inital_mean_length` and `inital_length_cv`.

```
@process Recruitment
type recruitment_beverton_holt
categories immature mature
proportions 1.0 0.0
r0 500000
steepness 0.75
ssb_offset 1
inital_mean_length 10
inital_length_cv 0.40
ssb SSB_derived_quantity
```

The property `ssb_offset` has to be manually specified.

YCS (YCS_y) The *YCS* parameter is reference by the recruited year. The recruited year is the year when a year class or age-cohort enter the partition. The recruited year differs from the spawning event year defined in Equation (5.5). This is a shift away from CASALs terminology which used `ycs_year` and is equivalent to the spawning event year. Standardisation years are are also now expressed as recruited years. This will differ from Casal2 versions before August 2022 and CASAL models. From August 2022 we deprecated the commands `ycs_values`, `ycs_years`, and `standardised_ycs_years`. These were replaced with `recruitment_multipliers` and `standardise_years`.

This year reference is important when defining `@estimate`, `@project`, and `@time_varying` blocks for the `recruitment_multipliers` parameter. 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 subcommand `standardise_years` is defined. The model parameter `recruitment_multipliers` is a vector $\mathbf{Y}$, covering the years from `start_year` to `final_year`. The resulting standardised recruitment multipliers are calculated as $YCS_i = Y_i/\bar{\mathbf{Y}}$, where the mean is calculated over the user-specified years `standardise_years`.

An alternative to "standardisation" is to constrain the *YCS* parameters using the simplex transformation (see Section 9). This is thought to have estimation benefits over the "standardisation" as priors can be applied to the "free" (estimable) parameters (Y_i).

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

where S is the set of years from `standardise_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_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*.

In the example given above, *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
...            #subcommand above
standardise_years 1995:2004
recruitment_multipliers 0.65 0.87 1.6 1.13 1.02 0.38 2.65 1.35 1 1 1 1 1
```

5.3.2 Mortality

There are 4 types of mortality processes available in Casal2length based models, plus the tag release processes that can also cause mortality (See Tagging Section 5.3.4):

- constant rate,
- constant exploitation,
- instantaneous, and
- disease.

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

```
# A process with label NaturalMortality
@process NaturalMortality
type          mortality_constant_rate
categories    male female
relative_m_by_length One One
m             0.2 0.2
```

The total number of individuals removed from a category

$$D_{j,t} = \sum_l N_{l,j,t}[1 - \exp(-S_{l,j}M_{l,j}p_t)] \tag{5.7}$$

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

The mortality rate process requires the specification of the mortality-by-length curve which is specified using a selectivity. To apply the same mortality rate over all length bins in a category, use a selectivity defined as $S_{l,j} = 1.0$ for all lengths l in category j

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

Length-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 5.9)

```
@selectivity length_specific_M
type double_exponential
x0 12
x1 1
x2 37
y0 0.182154
y1 1.43768
y2 1.57169
alpha 1.0

@process      NaturalMortalityByLength
type          mortality_constant_rate
categories    male female
relative_m_by_length length_specific_M length_specific_M
m             1.0 1.0
```

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

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

```
# A process with label IncidentalMortality
@process IncidentalMortality
type          mortality_constant_exploitation
categories    male female
relative_u_by_length One One
u             0.2 0.2
```

The total number of individuals removed from a category

$$D_{j,t} = \sum_l N_{l,j,t}S_{l,j}U_{l,j}p_t \tag{5.8}$$

where $D_{j,t}$ is the total number of removals in category j in time step t, $N_{l,j,t}$ is the number of individuals in category j of length bin l in time step t, $S_{l,j}$ is the selectivity value for length bin l in category j, $U_{l,j}$ is the exploitation rate for category j for length bin l, and p_t is the proportion of the exploitation rates to apply in time step t.

The exploitation rate process requires the specification of the mortality-by-length curve which is specified using a selectivity. To apply the same mortality rate over all length bins in a category, use a selectivity defined as $S_{l,j} = 1.0$ for all lengths l in category j

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

Length-specific exploitation rates can also be applied.

Instantaneous mortality The instantaneous mortality process combines both natural mortality and fishing exploitation into a single process. This allows the simultaneous application of both natural mortality and anthropogenic mortality to occur across multiple time steps. This process accounts for half the natural mortality within a time step before calculating vulnerable biomasses for calculating exploitation rates. The remaining half of the natural mortality is taken after exploitation has been accounted for. The input for this process is catches and these can either be specified as biomasses or numbers (abundance). In fisheries models in Casal2 this is the most commonly used mortality process.

This process allows for multiple removal events, e.g., a fisheries model with multiple fisheries and/or fleets. A removal method can occur in one time step only, although multiple removals can be defined to cover events during the year.

The equations for instantaneous mortality are based on Pope's discrete catch equation, which assumes catch is known without error. Casal2 will try and take the exact catch specified in the input.

- An exploitation rate (a proportion) is calculated for each fishery, as the catch divided by the selected-and-retained abundance or biomass termed vulnerable biomass. Vulnerable biomass is calculated by accounting for half natural mortality ($M_{l,c}$) that occurs at time-step which is defined by the subcommand `time_step_proportions` and denoted by p_t,

$$U_f = \frac{C_f}{\sum_c \sum_l \bar{w}_{l,c} S_{f,l,c} n_{l,c} exp(-0.5 p_t M_{l,c})} ,$$

  where $S_{f,l,c}$ is the fishery selectivity for length bin l and category c, $\bar{w}_{l,c}$ is mean weight and $n_{l,c}$ numbers at length before applying fishing. The categories c are user defined for each fishery f, which are defined in the `table method` (see below for an example).

- The fishing 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

$$U_{f,obs} = max_{l,c}(\sum_k \sum_c S_{k,l,c} U_k)$$

  where the maximum is over all partition elements (length and categories) affected by fishery f, and the summation is over all fisheries k which affect these partition elements in the same time step as fishery f.

  In cases with a single fishery the fishing 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 $S_{f,l}$ 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

$$n'_{l,c} = n_{l,c} exp(-p_t M_{l,c}) \left[1 - \sum_f S_{f,l,c} U_f\right]$$

For example, to apply natural mortality of 0.20 across three time steps on both male and female categories, with two methods of removals (fisheries `FishingWest` and `FishingEast`) and their 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_proportions 0.42 0.25 0.33
relative_m_by_length One
categories male female
biomass true
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 the label from the `catch` or `method` table and continuing the example,

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

Disease mortality rate Disease mortality is a special, additional, mortality that is implemented to occur after natural and fishing mortality during a time step. This process removes fish from the partition, is applied to all areas, and can depend on sex/length class.

The partition is updated as follows

$$n'_{c,j} = n_{c,j} exp\{-t_y M_c S_{c,j}\} \tag{5.9}$$

where $n_{c,j}$ is the partition for category c and length class j before mortality, and $n'_{c,j}$ is after the process. t_y is an annual multiplicative scalar (estimable), M_c is the category specific mortality rate and $S_{c,j}$ is the selectivity.

```
@process DiseaseMortality
type mortality_disease_rate
disease_mortality_rate 1.0
selectivities DiseaseSel
categories OYS
year_effect 0.05 0.11 0.39 0.38 0.20
years 2000 2001 2002 2003 2004 2005
```

5.3.3 Transition By Category

The transition by category process moves individuals between categories. This process is used to specify transitions such as maturation (individuals move from an immature to mature state) and migration (individuals move from one area to another).

There is a one-to-one relationship between the ”from” category and the ”to” category, i.e., for every source category there is one target category only

$$N_{l,j} = N_{l,i} \times P_i \times S_{l,i} \tag{5.10}$$

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

To merge categories repeat the ”to” category multiple times.

For 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 length or length classes that are mature and thus move to the eastern area.

If you want to estimate the proportion parameter, the parameter is addressed using the the `to` category. For example using the above process the estimate for the proportion parameter would follow

```
@estimate proportion_male_spawning
parameter process[Spawning_migration].proportions{East.males}
...
```

The transition by category process can be (optionally) included within a mortality block, but note that this may result in negative or nonsensical derived quantities or observations in either the ”from” or the ”to” categories. If used within the mortality block, care should be taken to ensure that any derived quantities or observations are correctly specified.

5.3.4 Tagging

Tag release events (also known as mark-recapture events or tag-release events) allow Casal2 to incorporate tagging data into the model.

Tagging is a process that moves fish from the general population into specific “tag” categories. The aim is to get Casal2 to track these separately to generate expected recaptures or growth etc.

In addition to creating tag category of the partition, you will need to initialise the values by defining a tag-release event (otherwise they will always be zero). This process moves fish from the "untagged" category of the partition into a named category of the partition. You will need to define how many fish to move, and the year, time step, area, and stock. Also, you may need to define a penalty (see Section 6.8) to avoid parameter values which do not lead to enough fish being present in the population to allow for the number being tagged (although in cases where only a small proportion of the population is tagged, this is unlikely to be required).

The partition is then updated by moving N fish from the equivalent "untagged" category of the partition to the named tag category of the partition, where the numbers at length are defined by a vector of proportions by category. Note that Casal2 expects the vector of proportions to sum to 1 over all length bins.

Casal2 moves fish using a "rate" which relates to the penalty. All "untagged" categories represented by $\tilde{c}$ are multiplied by a selectivity to calculate total abundance available to be tagged denoted by $V_{c,j}$

$$V_{c,j} = \sum_{\tilde{c}} n_{\tilde{c},j} S_{c,j} \; \tilde{c} \in c$$

where $\tilde{c}$ are categories that are a subset of c which can be an accumulation of multiple categories. Users define the number of tags released denoted by N and the proportion by length denoted by $p_{c,j}$. See below for an example configuration file excerpt.

$$U_{c,j} = \frac{N p_{c,j}}{V_{c,j}}$$

If this rate is greater than the input subcommand `u_max`

$$U_{c,j} = \begin{cases} U_{c,j} \geq u_{max}, & u_{max} \text{ flag a penalty} \\ U_{c,j} < u_{max}, & U_{c,j} \end{cases}$$

Tagged fish are moved as follows

$$T_{c,j} = U_{c,j} V_{c,j}$$

```
@process tag_1996
type tagging
years 1996
from untagged.male
to 1996_3.male
initial_mortality 0
u_max 0.99
selectivities [type=constant; c=1]
penalty none
N 61
table proportions
year 20 21 22 23 24 25 26 27 28 29 30
1996 0.016 0 0.016 0 0.032 0 0 0.01 0.045 0.048 0.016
end_table
```

5.4 Derived quantities

Some processes require a population value derived from the population state as an argument. These values are `derived quantities`. Derived quantities are values calculated in a specified time step in every year,

and thus have a single value for each year of the model. The time within the time-step is at the end unless otherwise specified (using the `proportion_mortality` subcommand).

Derived quantities can be calculated as either abundance or biomass. Abundance-derived quantities are the sum over the specified categories (after applying a selectivity). Biomass-derived quantities are calculated similarly.

In addition, a derived quantity (e.g., of type `biomass_index`) can have random error, autocorrelation, and bias added (however, these should never be used as quantities for determining recruitment or other parameters in estimation or MCMCs). Additionally, a derived quantity of type `biomass_index` can be modified by a catchability (q) coefficient, for example, to reproduce a biomass index that may be obtained from a survey.

Derived quantities are also calculated during the initialisation phases. Therefore, the time step during each initialisation phase must be specified. If the initialisation time steps are not specified, the derived quantity will be calculated during the initialisation phases.

A common use of an derived quantities is as input into a stock-recruit relationship which requires an equilibrium biomass (B_0) and annual spawning stock biomass values (SSB_y) to calculate recruitment into the first length class. SSB_y is an derived quantity based on the mature biomass, usually at spawning time. Additionally a derived quantity of type `biomass_index` could be used to inform a harvest strategy simulation.

Derived quantities can be associated with a *time evaluation interval*; see Section 5.3.2 for more detail on mortality blocks. In this case, the point of calculation can be set to any point within the mortality block, e.g., when 75% of the deaths from natural mortality plus catch has occurred, which is based on interpolating between the start and end of the block as the partition is known at those points. Two methods are available: `weighted_sum` and `weighted_product`, and are defined as

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

For example, to define a biomass-derived quantity spawning stock biomass, *SSB*, calculated 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, the syntax is

```
@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
```

To generate a biomass index derived quantity for a harvest strategy calculation, generated 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 with variance (normally distributed with CV = cv) and autocorrelation (rho) the syntax is

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

```
distribution normal
cv 0.2
rho 0.1
bias 0.0
```

5.5 Growth

5.6 Growth Increment models

For length-based models the length structure of the partition has n_l length bins where l_i denotes length bin i with minimum length value denoted by lc_i and midpoint denoted as lm_i.

$$lm_i = 0.5\,(lc_i + lc_{i+1})$$

In length-based models a growth transition matrix describes the probability of fish moving from length bin l_j to length bin l_k. The growth increment models describe the expected mean length increment denoted by μ_i for fish in length bin l_i.

5.6.1 The 'none' model

All individuals have a mean length increment of 1 regardless of where they are in the length partition.

5.6.2 The Exponential model

$$\mu_i = g_\alpha \left(\frac{g_\beta}{g_\alpha}\right)^{\frac{(lm_i - l_\alpha)}{(l_\beta - l_\alpha)}} \tag{5.11}$$

5.6.3 The Basic model

$$\mu_i = g_\alpha + (g_\beta - g_\alpha)\frac{lm_i - l_\alpha}{l_\beta - l_\alpha} \tag{5.12}$$

5.6.4 Growth transition matrix

The probability of a fish in length bin i moving into length bin j is defined by the growth transition matrix. This matrix is defined by length bin midpoints (lm_i) mean increment μ_i a standard deviation and distribution assumption.

All growth increment models require a coefficient of variation (cv) and minimum standard deviation σ_{min}.

$$\sigma_i = max(\sigma_{min}, \mu_i cv)$$

The growth transition matrix in time step t denoted by $\mathbf{G^t}$is defined as follows

$$G^t_{i,j} = \begin{cases} 0.0\,, & \text{for } j < i \text{ no negative growth} \\ f(lc_{i+1}, \mu_i, \sigma_i)\,, & \text{for } i = j \\ 1.0 - \sum_{k=1}^{n_l-1} G^t_{i,k}\,, & \text{for } i = n_l \text{ \& plus group} \\ f(lc_{n_l+1}, \mu_i, \sigma_i) - \sum_{k=1}^{n_l-1} G^t_{i,k}\,, & \text{for } i = n_l \text{ \& no plus group} \end{cases} \tag{5.13}$$

where, $f(X, \mu, \sigma)$ is the cumulative density function defined by `distribution`. Currently only the Normal distribution is allowed.

5.7 Length-weight relationship

There are two length-weight relationships options. The first is the naive "no relationship" relationship, where the weight of an individual is always 1, regardless of length. The second relationship is the "basic" relationship, which is the standard length-weight relationship, $W = aL^b$.

5.7.1 The 'none' relationship

$$\text{mean weight} = 1 \tag{5.14}$$

5.7.2 Basic: the standard length-weight relationship

The mean weight $\bar{w}$ of an individual of length l is

$$\bar{w} = al^b \ . \tag{5.15}$$

This is used in length based models where l is the length bin midpoint.

Note: the scale of a can be specified incorrectly. If the catch is in tonnes and the growth curve is in centimetres, then a should convert a length in centimetres to a weight in tonnes. There are reports available that can be used to help check that the units specified are plausible (see Section 8).

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

5.8 Maturity, in models without maturing in the partition

When maturity is not an attribute (explicit category) in the partition, processes may still depend on maturity. You must then make the assumption that the proportion of mature fish in each element is defined by a selectivity ogive. This approximation is used by derived quantities (Section 5.4). Selectivity ogives are allowed to vary over time with the time-varying class (Section 5.10)

5.9 Selectivities

Selectivity is a term used in Casal2 to mean an ogive in both age and length based models. They can be used to specify the selection curve for a fishery or observation (Section 6) or to modify the effects of processes on the partition, e.g., migration rates by age (Section 5).

There are a number of different parametric forms, including logistic and double normal curves. Selectivities are defined in command block `@selectivity[label]`, where the unique label of the selectivity is used by observations and processes to specify which selectivity to apply.

In length-based models Casal2 will use length midpoints when calculating selectivities.

There are a number of different parametric forms, including logistic and double normal curves. Selectivities are defined in command block `@selectivity[label]`, where the unique label of the selectivity is used by observations and processes to specify which selectivity to apply.

Many selectivities can be forced to apply to ages or lengths from a specified age (or length), i.e., a logistic selectivity can be defined with

```
@selectivity trawlSel    # label for the trawl fishery selectivity
type      logistic       # type of curve
a50       4.4            # length class at 50% selection
ato95     1.5            # interval (yr) from a50 to the length
                         #   class at 95% selection
```

```
beta      2              # minimum length class selected, so that individuals with
                         #   length < beta have selectivity = 0
```

For some selectivities, the function values for some choices of parameters can result in numeric overflow or underflow errors (i.e., the number calculated from parameter values is either too large or too small to be well represented). Casal2 implements range checks on some parameters to test for these errors before calculating 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 selectivity options are:

- Constant (Section 5.9.1)
- Knife-edge (Section 5.9.2)
- All values (Section 5.9.3)
- All values bounded (Section 5.9.4)
- Increasing (Section 5.9.5)
- Logistic (Section 5.9.6)
- Inverse logistic (descending logistic?) (Section 5.9.7)
- Logistic producing (Section 5.9.8)
- Double normal (Section 5.9.9)
- Double normal plateau (Section 5.9.10)
- Double normal stock synthesis (Section 5.9.11)
- Double exponential (Section 5.9.12)
- Compound-left (Section 5.9.13)
- Compound-right (Section 5.9.14)
- Compound-middle (Section 5.9.15)
- Compound-all (Section 5.9.16)
- Multi-selectivity (Section 5.9.17)

See Figure 5.2 for example plots of the selectivities (p. 49).

5.9.1 constant

The constant selectivity is constant power function ($ax^b + c$) for all age/lengths greater than β. For $x < \beta$, the selectivity is zero.

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ ax^b + c, & otherwise \end{cases} \tag{5.16}$$

For a simple constant selectivity (i.e., where $f(x) = 1$ for all length bins set a and $b = 0$ and $c = 1$).

To implement an inverse length selectivity (to use, for example, as a selectivity for natural mortality (M) where M is inversely proportional to length — see Lorenzen et al. (2022)), use $a = 1$, $b = -1$, and $c = 0$).

The constant selectivity has estimable parameters a, b, and c.

Input fragment:

```
type constant
a    0.0
b    0.0
c    1.0
beta 0.0 # the default is 0.0
```

5.9.2 knife_edge

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{5.17}$$

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

Input fragment:

```
type  knife_edge
e     8
alpha 0.5
```

5.9.3 all_values

$$f(x) = V_x \tag{5.18}$$

The all-values selectivity has estimable parameters V_{low}, V_{low+1} $\ldots V_{high}$. The selectivity value for each age/length class must be set.

5.9.4 all_values_bounded

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \tag{5.19}$$

The all-values-bounded selectivity has non-estimable parameters L and H. The estimable parameters are V_L, V_{L+1} $\ldots V_H$. Selectivity values for each age/length class from $L \ldots H$ must be set.

Selectivities `all_values` and `all_values_bounded` can be included in additional priors using the syntax

```
@selectivity maturity
type all_values
v 0.001 0.1 0.2 0.3 0.4 0.3 0.2 0.1

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

5.9.5 increasing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \leq x \leq H \\ f(\alpha), & \text{if } x \geq H \end{cases} \tag{5.20}$$

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

Input fragment:

```
type  increasing
l     3
h     7
v     0.2 0.3 0.4 0.5 0.6
```

5.9.6 `logistic`

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ \alpha/[1+19^{(a_{50}-x)/a_{to95}}], & \text{otherwise} \end{cases} \tag{5.21}$$

The logistic selectivity has estimable parameters a_{50} and a_{to95}. α is a scaling parameter, with default value of $\alpha = 1$. β is the minimum age/length to which the selectivity applies.

The logistic selectivity has values 0.5α at $x = a_{50}$ and 0.95α at $x = a_{50} + a_{to95}$. For $x < \beta$, the selectivity is zero.

5.9.7 `inverse_logistic`

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ \alpha - \alpha/[1+19^{(a_{50}-x)/a_{to95}}], & \text{otherwise} \end{cases} \tag{5.22}$$

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

The inverse logistic selectivity has values 0.5α at $x = a_{50}$ and 0.95α at $x = a_{50} - a_{to95}$. For $x < \beta$, the selectivity is zero.

Input fragment:

```
type  inverse_logistic
a50   4
ato95 1
alpha 0.5 # the default is 1.0
beta  0.0 # the default is 0.0
```

5.9.8 `logistic_producing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ \left(\lambda(x) - \lambda(x-1)\right) / \left(1 - \lambda(x-1)\right), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \tag{5.23}$$

The logistic-producing selectivity has non-estimable parameters L and H. The estimable parameters are 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 either movement or maturity. In such a model, a logistic-producing selectivity will, in the absence of other influences, make the proportions moved or mature follow a logistic curve with parameters a_{50} and a_{to95}.

Input fragment:

```
type  logistic_producing
l      2
h      8
a50    4
ato95  1
# alpha 1.0
```

CASAL's implementation of this selectivity adds the following checks.

for(i in selectivity_bins)
$if((a_{50}-i)/a_{to95} < -5.0))$
$selectivity[i] = 1$

for(i in selectivity_bins)
$if((a_{50}-i)/a_{to95} > 5.0))$
$selectivity[i] = 0$

Casal2 does not have these checks, so when you plot selectivities they may look different at the edges.

5.9.9 double_normal

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \tag{5.24}$$

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 $x = a_1 + s_R$. For $x < \beta$, the selectivity is zero.

Input fragment:

```
type  double_normal
mu       6   # age/length class at switch over from left to right normal curves
             #  = mean for both normal curves
sigma_1  1   # standard deviation for left normal
sigma_2 10   # standard deviation for right normal
# alpha  1.0
# beta   0.0
```

5.9.10 double_normal_plateau

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ \alpha 2^{-[(x-a1)/\sigma_L]^2}, & \text{if } x \leq a1 \\ \alpha, & \text{if } a1 \leq x \leq a1 + a2 \\ \alpha 2^{-[(x-(a1+a2))/\sigma_R]^2}, & \text{if } x \geq a1 + a2 \end{cases} \tag{5.25}$$

The `double_normal_plateau` ogive has estimable parameters $a1$, $a2$, σ_l, σ_r, and α.

When $\alpha = 1$ and $a2 = 0$, it is identical to the `double_normal`, and otherwise follows a double normal form with values α at $a1 \leq x \leq a1 + a2$, and $0.5x\alpha$ at $x = a1 - \sigma_l$ or $x = a1 + a2 + \sigma_r$. For $x < \beta$, the selectivity is zero.

Input fragment:

```
type  double_normal_plateau
a1       6
a2       2
sigma_1  1   # standard deviation for left normal
sigma_2  10  # standard deviation for right normal
# alpha 1.0
# beta  0.0
```

5.9.11 double_normal_stock_synthesis

Double normal with defined initial and final selectivity values which is based on the Stock Synthesis 3 implementation. The `ascending` and `descending` are expected in log space (This should be taken out and dealt with by the parameter transformation class in future releases).

It is common to estimate the following parameters `peak`, `width`, `ascending` and `descending`. `y1` can be explored, but can be difficult to estimate, as generally this represents the age or length categories that are not well observed.

Input fragment:

```
type  double_normal_stock_synthesis
peak  7.5 # age or length for the plateau, should be between L and H
y0  -10  # selectivity at min-age or first length bin see below for units
y1   0.5 # selectivity at max-age or last length bin see below for units
descending    # log(age or length) of descending limb (shape of right hand side)
ascending    # log(age or length) of ascending limb (shape of left hand side)
width   3     # width of plateau
L   1     # first length bin
H  10    # last age bin
#alpha 1.0
```

The parameter values `y0` and `y1` are transformed by the selectivity class as follows

$$f(x) = \frac{1}{1+exp(-1.0x)}$$

This is to ensure the values stay between 0 and 1. The down side is that the starting values are a little abstract. The rule of thumb is small numbers, e.g., -10, will result in selectivity values close to zero and large values, e.g., 10, will result in selectivity values close to one.

5.9.12 double_exponential

$$f(x) = \begin{cases} 0, & \text{if } x < \beta \\ \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \leq x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{5.26}$$

The double-exponential selectivity has non-estimable parameters x_1 and x_2. The estimable parameters are x_0, y_0, y_1, and y_2. α is a scaling parameter, with default value of $\alpha = 1$.

This selectivity curve 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). For $x < \beta$, the selectivity is zero.

Input fragment:

```
type  double_exponential
x0    15   # age/length at middle point
y0    0.1  # selection at x0; here a minimum --> U shape
x1    1    # left point
y1    0.5  # selection at x1
x2    30   # right point
y2    0.8  # selection at x2
# alpha 1.0
# beta  0.0
```

5.9.13 compound_left

The compound left selectivity was used in some oyster stock assessments but was not documented in CASAL user manual.

$$
\begin{aligned}
y_1 &= \frac{(1-a_{min})}{\left(1+19^{(a_{50}-x)/a_{to95}}\right)} + a_{min} \\
y_2 &= 1.0 - \frac{1}{\left(1+19^{(left_{mu}+to\_right_{mu}-x)/\sigma}\right)} \\
f(x) &= y_1 y_2
\end{aligned}
$$

5.9.14 compound_right

The compound right selectivity was used in some oyster stock assessments but was not documented in CASAL user manual.

$$
\begin{aligned}
y_1 &= \frac{(1-a_{min})}{\left(1+19^{(a_{50}-x)/a_{to95}}\right)} + a_{min} \\
y_2 &= \frac{1}{\left(1+19^{(left_{mu}+to\_right_{mu}-x)/\sigma}\right)} \\
f(x) &= y_1 y_2
\end{aligned}
$$

5.9.15 compound_middle

The compound middle selectivity was used in some oyster stock assessments but was not documented in CASAL user manual.

$$
\begin{aligned}
y_1 &= \frac{(1-a_{min})}{\left(1+19^{(a_{50}-x)/a_{to95}}\right)} + a_{min} \\
y_2 &= \frac{1}{\left(1+19^{(left_{mu}+to\_right_{mu}-x)/\sigma}\right)} \\
y_3 &= 1.0 - \frac{1}{\left(1+19^{(left_{mu}+to\_right_{mu}-x)/\sigma}\right)} \\
f(x) &= y_1 y_2 y_3
\end{aligned}
$$

5.9.16 compound_all

The compound all selectivity was used in some oyster stock assessments but was not documented in CASAL user manual.

$$
f(x) = \frac{(1-a_{min})}{\left(1+19^{(a_{50}-x)/a_{to95}}\right)} + a_{min}
$$

5.9.17 multi_selectivity

This selectivity allows users to configure models that have different selectivity functions applied in different years. For each year of the model, a selectivity is defined using a label of a selectivity defined elsewhere in the input configuration files. Each year can be either unique or else blocks of years can define a different selectivity. For example,

```
@selectivity fishery_selectivity
type multi_selectivity
years 1990:2000 2001:2010
selectivity_labels early_2000s_sel * 10 post_2000_sel * 9
default_selectivity post_2000_sel

@selectivity early_2000s_sel
type logistic
a50 4
ato95 2.3

@selectivity post_2000_sel
type double_normal
mu 6
sigma_l 2
sigma_r 23
```

For missing years, the `default_selectivity` is applied. Missing years in projections use the `projection_selectivity` if the subcommand has been supplied, otherwise the `default_selectivity` will be used for those missing years.

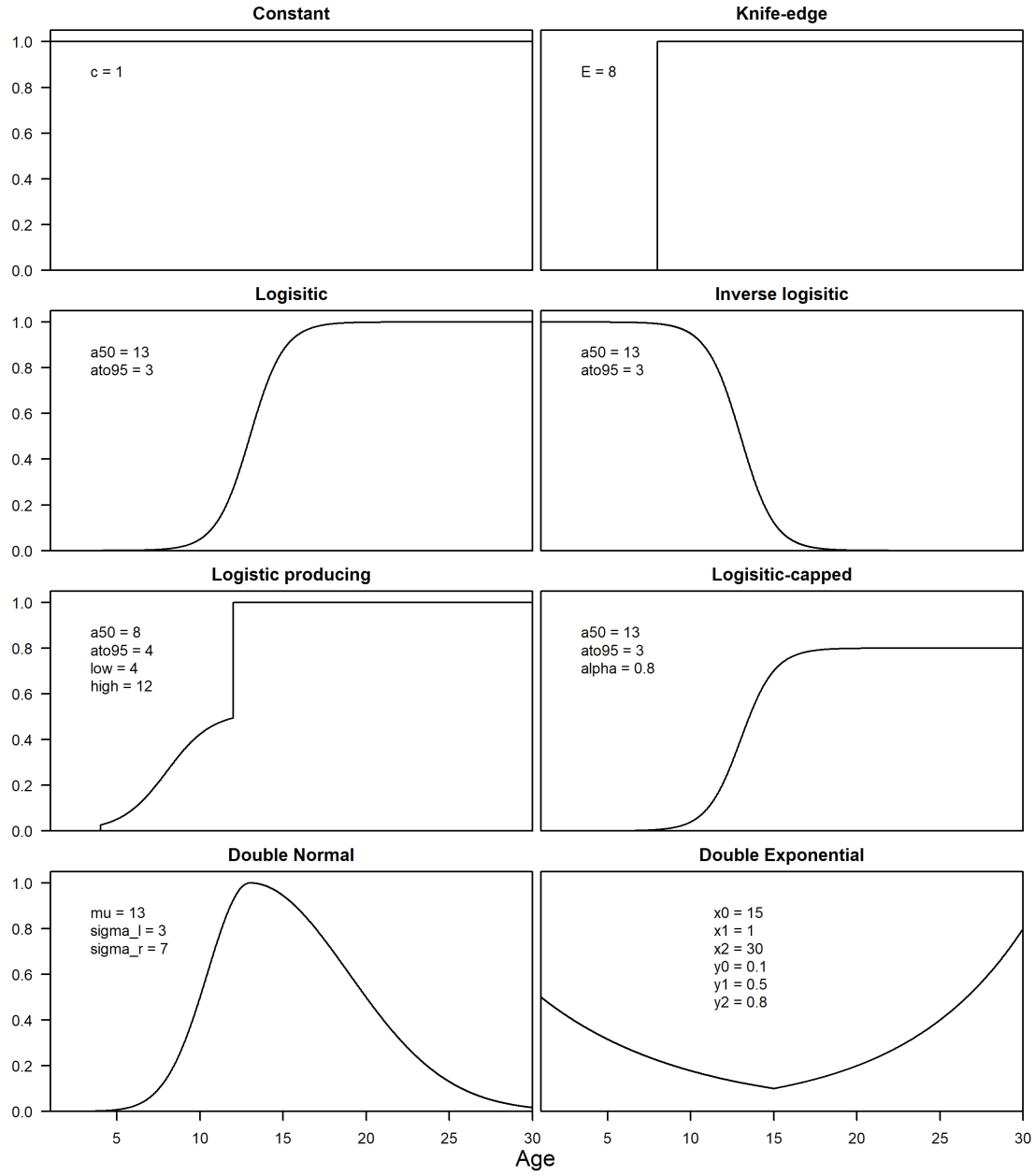

Figure 5.2: Examples of the selectivities

5.10 Time-varying parameters

Any parameter can be varied annually for blocks of years or in specific years within the model run. 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. The value used in the configuration file, input parameter file, or trialled value during estimation should be applied during initialisation phases.

Method types for a time-varying parameter are:

- `constant`,
- `exogenous`,
- `linear`,
- `annual_shift`,

- random_walk, and
- random_draw.

This option allows for a parameter to be fixed in a year, or be the result of a deterministic or stochastic process. Note that the stochastic time-varying methods (e.g., random_walk and random_draw) are intended for simulations or projections — they should not be used in estimation as they utilise random numbers to generate parameter values.

To implement a hierarchical model using the time-varying functionality, use MCMC estimation as a way to calculate the integral which is required to obtain unbiased estimates. Here, the prior parameter values need to be estimated using hyper-priors. In an MCMC context, a Gibbs sampler is assumed. That is, every draw is from a conditional distribution and so every draw is a candidate value.

When allowing time-varying parameters (such as in catchability coefficients or selectivities parameters), a model is given freedom to more closely match the observed data. Time-varying parameters can be used to allow the mean or shape parameters of selectivities to change between years, and potentially as a function of a linked variable.

An example of this is in the New Zealand hoki stock assessment where the μ and a_{50} parameters are allowed to shift depending on when the fishing during the 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 Examples/2stock directory, implemented at line: 382 in the population.csl2 file.

5.10.1 constant

This option allows a parameter to have an different value during specified years than the rest of the model run. This value can be estimated.

To allow survey catchability to be different in the year block 1975 to 1988 from the rest of the series we write:

```
@time_varying q_time_var
type          constant
parameter     catchability[survey_q].q
years         1975:1988
values        0.001 # the same for all years
```

To estimate catchability for 1975 and 1976, use the following:

```
@estimate q_time_var
type uniform   #prior
parameter time_varying[q_time_var].values{1975:1976}
lower_bound 1e-6 1e-6
upper_bound 2    2
```

To make the catchability be same over the year block we need to estimate it for one year (say 1975) and use the *same* subcommand to make the others take the same value

```
@estimate q_time_var
type uniform
parameter time_varying[q_time_var].values{1975}
same      time_varying[q_time_var].values{1976:1988}
lower_bound 1e-6
upper_bound 2
```

Caution: do not estimate both the actual parameter and its time-varying counterpart, as the time-varying value will overwrite the actual parameter making the actual value unidentifiable.

5.10.2 `linear`

Parameters are shifted based on a linear trend, with a given slope and intercept. An example of this is an exploitation selectivity parameters that may increase or decrease between years based a simple linear trend.

$$\delta_y = aE_y + b \tag{5.27}$$

$$\theta'_y = \theta_y + \delta_y \tag{5.28}$$

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 slope parameter and b is the linear trend intercept.

5.10.3 `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.

$$\delta_y = a(E_y - \bar{E}) \tag{5.29}$$

$$\theta'_y = \theta_y + \delta_y \tag{5.30}$$

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).

5.10.4 `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

$$\bar{\theta}_y = \frac{\sum_y^Y \theta_y}{Y} \tag{5.31}$$

$$\theta'_y = a\bar{\theta}_y + b\bar{\theta}_y^2 + c\bar{\theta}_y^3 \tag{5.32}$$

5.10.5 `random_walk`

A random deviate drawn from a standard normal distribution is added to the previous year's value. This option has an estimable parameter σ_p for each time-varying parameter p. For reproducible modelling when using stochastic functionality, set the random seed (see Section 3.3).

```
@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 safeguard against the application of a random deviate resulting in parameter values which cause the model to fail, i.e., generates NA or INF values. To avoid this, specify an @estimate block even though the parameter is not actually being estimated; see the example syntax below.

A constraint whilst using this functionality is that a parameter cannot be less than 0.0. If it is then 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 configuration will insure the random walk time-varying process will set the any new candidate values within the lower and upper bound of the @estimate block.

5.10.6 random_draw

A random deviate drawn from a standard normal distribution used. This option has an estimable parameter σ_p for each time-varying parameter p. For reproducible modelling when using stochastic functionality, set the random seed (see Section 3.3).

```
@time_varying q_time_var
type          random_draw
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 safeguard against the application of a random deviate resulting in parameter values which cause the model to fail, i.e., generates NA or INF values. To avoid this, specify an @estimate block even though the parameter is not actually being estimated; see the example syntax below.

A constraint whilst using this functionality is that a parameter cannot be less than 0.0. If it is then 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 configuration will insure the random draw time-varying process will set the any new candidate values within the lower and upper bound of the @estimate block.

5.11 Equation parser

Casal2 has an equation parser, which is currently implemented in Projections (Section 5.12), Derived quantities (Section 5.4), and Reports (Section 8).

Examples of syntax for implementing the equation parser are below. For more information on the parser, see https://github.com/nickgammon/parser/blob/master/parser.cpp

```
equation process[Recruitment].r0 * 2 #double the recruitment
```

mathematical functions such as `sqrt()`, `log()`, `exp()`, `cos()`, `sin()`, and `tan()` can be used

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

exponents can be used with `pow()`

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

the absolute value of an equation using `abs()`

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

`if-else` statements can be used

```
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, more complex syntax

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

Only single values can be referenced, so an equation cannot be applied to a vector, e.g., `process[Recruit].recruitment_multipliers{1974:1980}` cannot be referenced. More information on which parameters can be included in the equation parser is available (Section 14.1.3). Any subcommand that has a `type estimable` could be referenced with the equation parser.

Note: the equation parser will not catch all user configuration errors, such as checking whether a parameter that exists in the system has been populated when it is required.

For example, the wrong year could be misspecified in the case of removals in year y which is based on the state of the population in year $y-1$

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

This example is a valid equation but it will have nonsensical results, since a value for 2020 is to be calculated using values for 2015. Although the equation parser adds flexibility, it is easy to incorrectly specify equations.

5.12 Projections

Projections are model runs, using a set of parameters, that are used to project the model forward in time to predict a future state.

`casal2 -f` *n*

to project each set of parameter values *n* times into the future.

A set of parameter values from the file specified with the argument -i or -I must be supplied. Projections are repeated for each parameter set (i.e., each line of data in the free parameter file) *n* times. Typically, the MCMC sample output will be used for projections using the output `samples` file with -i.

5.12.1 Specifying projections

Given a set of estimated parameter values from a *-e* or a MCMC run, the model can be projected. Projection years are after the model run years, and are defined in the `@model` command block using the `final_projection_year` subcommand, i.e., projection years are `final_year + 1` through to `final_projection_year`.

Parameter values for the projected years can be specified in a stochastic way or fixed at some value (the default is the estimated value if the parameter is not time-varying) and these are specified in the `@project` block,

```
@project Future_ycs  # label
type       lognormal_empirical  # which method to use
parameter  process[Recruitment].ycs_values
years      2012:2016
multiplier 1
... # any other parameters
```

The subcommands `years` and `parameter` are common to all projection methods. Subcommand `years` specifies the years to apply the new values to for the parameter in `parameter`. Note that the years can be before the *final_year*, e.g., it is normal to vary the last few recruitment multipliers (YCS) in a projection run because they are usually poorly estimated or they have been set to 1.

The `type` subcommand gives the method to use to generate new parameter values.

The argument `multiplier` is a constant (which may be set the same for all years or a value given for each year) which is multiplied with the projected value (for all but the multiple values method) after it has been generated.

Casal2 allows any estimable parameter to be specified in a `@project` block and then varied from the estimated value in a projection. The available projection types for these parameters include:

- constant
- lognormal
- empirical-lognormal
- empirical re-sampling
- harvest strategy constant catch
- harvest strategy constant U
- harvest strategy ramp U

Casal2 has no default projection properties for parameters that are specified by year, e.g., recruitment multiplier parameters, time-varying parameters, or future catches. For these, projections must have a `@project` command block. For example, Casal2 will produce errors if run in projection mode without a `@project` block for the `recruitment_multipliers` parameter being specified.

Future catches are also specified in a `@project` block, one for each fishery (see 5.12.7 for examples). Here, a fishery is reference in the `parameter` subcommand with the *method_* fragment to identify it,

`process[block label].method_[fisheries label]`,

For a process called `Fishing` that has three fisheries defined, it would be `process[Fishing].method_pot` to specify the fishery labelled *pot*.

The Casal2 command to run the model in projection mode is `Casal2 -f 1`. 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. For projections, a free parameter file must be supplied using `-i/-I`

The `--tabular` flag should be used when running projections after a Bayesian analysis. This option will output a tabular report (see Section 8.31) which can then be analysed in **R**.

An example of the command line evocation is

```
casal2 -f 1 -i samples.1 --tabular > projection.log
```

where `samples.1` is output from a MCMC run, one parameter set per row, which will give one projection per row, and `projection.log` will contain one row for each MCMC run in each of the reports specified.

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.

5.12.2 Projection method of type Constant

A parameter can either be fixed during all projection years or specified individually for each projection year using `constant` projection command. 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].recruitment_multipliers
years 2012:2016
values 1.0 2.0 1.0 2.0 0.5 # one value per year. Alternatively, a single value means
                           that all years use the same value
```

5.12.3 Projection method of type Multiple Values

Users can specify a set of values for each projection year for each row in the `-i` or `-I` file using the `multiple_values` projection command. This gives users flexibility in specifying a range of bespoke values during projections and including uncertainty in the projected values. See below for an example on how to configure this.

```
@project future_disease_rates
type multiple_values
parameter process[dtransition].proportions{disease}
years 2024:2026
table values
# 2024 2025 2026
  0.1  0.2  0.3
  0.3  0.4  0.5
end_table
```

5.12.4 Projection method of type Empirical Sampling

Parameters that have time components associated with them can be sampled uniformly with replacement over a range of years and used as values for the projected years with the `@empirical_sampling` projection command. The year range to sample from is between `start_year` and `final_year`:

```
@project Future_ycs
type empirical_sampling
parameter process[Recruitment].standardised_recruitment_multipliers
years 2012:2016
start_year 1988 # re-sample from estimated values
final_year 2008 # from 1988 to 2008 inclusive
multiplier 1
```

Note that if projecting `recruitment_multipliers` with the Haist parameterisation of recruitment multipliers, the *unstandardised* values will be used. This may cause projections to be other than what was

intended unless these (by chance) have a mean of one. Note that Casal2 will not warn nor report an error if this is done.

In this specific case, project the `standardised_recruitment_multipliers` instead, as these will have a mean of one (by definition). If using `standardised_recruitment_multipliers`, note that Casal2 versions before v24.01 may not implement this correctly and a more recent version is recommended.

Additionally, if not using a transformation (e.g., the simplex transformation for recruitments), a vector average penalty could be applied (and the resulting recruitment multipliers checked) to ensure the average recruitment multiplier is close to one over some pre-defined range. Note that this is recommended in any case, as the prior on recruitments is applied to the unstandardised values when the Haist parameterisation is used.

5.12.5 Projection method of type Lognormal

The `@lognormal` projection command draws parameters from a Gaussian (Normal) distribution in log space and exponentiated to result in the lognormal distribution

$$X_p = exp(\varepsilon_p - \sigma^2/2) \tag{5.33}$$

where $\varepsilon_p \overset{iid}{\sim} N(\mu, \sigma)$ and X_p is the projected value for parameter X, and μ and σ are the mean and standard deviation on the log scale.

An example of applying this process to draw future year class parameters from a lognormal distribution with mean 1 and standard deviation 0.8

```
@project Future_ycs
type lognormal
parameter process[Recruitment].recruitment_multipliers
years 2012:2016
mean 0 # mean 1 on un-transformed scale
sigma 0.8 # log scale
multiplier 1
```

5.12.6 Projection method of type Lognormal-Empirical

`lognormal_empirical`

The `@lognormal_empirical` command applies a lognormal draw as in the `Lognormal` method above and specifies a year range from which the standard deviation of the distribution is calculated. Then equation (5.33) is used to generate future values with a specified μ and empirically calculated σ,

```
@project Future_ycs
type lognormal_empirical
parameter process[Recruitment].recruitment_multipliers
years 2012:2016
mean 0
start_year 1988 # range of years to take the
final_year 2008 # values for \math{\sigma}
multiplier 1
```

5.12.7 Projecting Catches

Specifying catches in a projection may require a slightly different format from the options identified above, as catches are usually specified in a table format. Here the table name (i.e., method) needs to be appended to specify the parameter to project. This uses the syntax `block_type[block_label].method_fishinglabel`.

For example, to project the `fishingPot` catches that are in the following table

```
# fishing process
@process Fishing
type mortality_instantaneous
m 0.17*6
time_step_proportions 1
relative_m_by_age One*6   # For age based model.
                          # For length-based models use relative_m_by_length
categories *
table catches
year  fishingPot other Recreation
1900  0          0     0
1901  13.2       0     22.9
1902  26.4       0     23.5
1903  39.6       0     24
end_table
```

use the parameter label `process[Fishing].method_fishingPot`, i.e.,

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

5.12.8 Projection method of type Harvest Strategy Constant Catch

The `@harvest_strategy_constant_catch` command applies a harvest strategy from a biomass index (i.e., a derived quantity). The harvest strategy is a constant catch method where the catch is updated according to a proportional change in the biomass index.

The biomass index is a derived quantity. A derived quantity of `type=biomass_index` is recommended as this will allow the introduction of random error, autocorrelation, and bias into the index when calculating the catch from the exploitation rate.

The constant catch harvest strategy applies a catch equal to `alpha` times the proportional change in the biomass index (scaled by `multiplier`), where the `biomass_index` is the value of the derived quantity from the current year less the `biomass_index_lag_years`. It will update the catch that is applied according to the frequency defined by `update_frequency_years`. The catch will only be applied if the change is greater than `min_delta`, and the maximum change that can be applied is equal to `max_delta` (where a value of zero is interpreted as no maximum).

The harvest strategy rule is only applied in the years in the projection period from `first_year`. Before `first_year` the `current_catch` is applied.

```
@project Future_ycs
type      harvest_strategy_constant_catch
parameter process[Instantaneous_Mortality].method_Sub_Ant_F
years     2012:2016
biomass_index SSB
catch 963.9
alpha 1.0
min_delta 0.1
max_delta 0.3
update_frequency_years 3
biomass_index_lag_years 2
current_catch 963.9
```

```
first_year 2009
multiplier 1
```

5.12.9 Projection method of type Harvest Strategy Constant U

The `@harvest_strategy_constant_u` method applies a harvest strategy from a biomass index (i.e., a derived quantity) and an exploitation rate. The harvest strategy is a constant U method where the catch is updated according to constant exploitation rate (U).

The biomass index is a derived quantity. A derived quantity of `type=biomass_index` is recommended as this will allow the introduction of random error, autocorrelation, and bias into the index when calculating the catch from the exploitation rate.

The constant U harvest strategy applies a catch equal to `U` times the biomass index (scaled by `multiplier`), where the `biomass_index` is the value of the derived quantity from the current year less the `biomass_index_lag_years`. It will update the catch that is applied according to the frequency defined by `update_frequency_years`. The catch will only be applied if the change is greater than `min_delta`, and the maximum change that can be applied is equal to `max_delta` (where a value of zero is interpreted as no maximum).

The harvest strategy rule is only applied in the years in the projection period from `first_year`. Before `first_year` the `current_catch` is applied.

```
@project Future_ycs
type      harvest_strategy_constant_u
parameter process[Instantaneous_Mortality].method_Sub_Ant_F
years     2012:2016
biomass_index SSB
u 0.15
min_delta 0.1
max_delta 0.3
update_frequency_years 3
biomass_index_lag_years 2
current_catch 963.9
first_year 2009
multiplier 1
```

5.12.10 Projection method of type Harvest Strategy Ramp U

The `@harvest_strategy_ramp_u` method applies a harvest strategy using a ramp rule. The harvest strategy is an exploitation rate $\hat{U}$ where the catch is updated according to a ramp rule using `u` and the `reference_index` to calculate $\hat{U}$, with this then multiplied by the `biomass_index` to determine the catch to apply.

The biomass index and reference index are derived quantities. Derived quantities of `type=biomass_index` is recommended as this will allow the introduction of random error, autocorrelation, and bias into the index when calculating the catch from the exploitation rate. The `biomass_index` and `reference_index` can be the same derived quantity.

The ramp U harvest strategy calculates the $\hat{U}$ to be applied from a ramp rule defined by a vector of `u`'s and a equal length vector of `reference_points`. Here, the value of $\hat{U}$ is a linear interpolation of the values of `u` at the inflection points defined by the `reference_points`.

Both the `reference_index` and `biomass_index` are derived quantities where their value is equal to the value of the current year less the `biomass_index_lag_years`. The `reference_index` is then divided by its value from the initialisation phase (i.e., is a proportion). Note that if a lognormal distribution is assumed for `reference_biomass`, the ratio of the initial to current biomass may need to be adjusted for the bias in the expectation. This can be applied using `bias_adjustment=true`.

The ramp rule harvest strategy applies a catch equal to $\hat{U}$ times the biomass index (scaled by `multiplier`). It will update the catch that is applied according to the frequency defined by `update_frequency_years`. The

catch will only be applied if the change is greater than `min_delta`, and the maximum change that can be applied is equal to `max_delta` (where a value of zero is interpreted as no maximum).

In addition, the value of $\hat{U}$ that is used is equal to the lowest value of `u` when the reference index is less than the first value of `reference_points`, and equal to the highest value of `u` when the reference index is greater than last value of `reference_points`. Otherwise, $\hat{U}$ is derived using linear interpolation between inflection points of the reference index.

The harvest strategy rule is only applied in the years in the projection period from `first_year`. Before `first_year` the `current_catch` is applied.

```
@project Future_ycs
type      harvest_strategy_ramp_u
parameter process[Instantaneous_Mortality].method_Sub_Ant_F
years     2012:2016
biomass_index SSB
reference_index SSB
u                0.00 0.00 0.10 0.25
reference_points 0.00 0.10 0.20 0.40
min_delta 0.1
max_delta 0.3
update_frequency_years 3
biomass_index_lag_years 2
current_catch 963.9
first_year 2009
multiplier 1
```

6 The estimation section: estimation methods and parameters

The command and subcommand syntax for the estimation section is given in Section 10.

6.1 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 6.2)
2. Define the parameters to be estimated (the free parameters, see Section 6.3)
3. Calculate a point estimate, i.e., the maximum posterior density estimate (MPD) (see Section 6.4)
4. Calculate a posterior profile on selected parameters, i.e., for each of a series of values of a parameter, minimise the objective function, allowing the other estimated parameters to vary (see Section 6.5)
5. Generate MCMC samples from the posterior distribution (see Section 6.6)
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 6.4)

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, the assumed priors, and the penalties. See the observation section for a description of the observations, likelihoods, priors, and penalties.

6.2 The objective function

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

$$Objective(p) = -\sum_i \log\left[L\left(\mathbf{p}|O_i\right)\right] - \log\left[\pi\left(\mathbf{p}\right)\right] \tag{6.1}$$

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

The contribution to the objective function from the likelihood components is described in Section 7.2. In addition to likelihoods, priors (see Section 6.7) and penalties (see Section 6.8) are components of the objective function. Note that if the priors are specified as uniform, then the prior contribution is zero and the optimisation is now a penalised likelihood and not Bayesian.

Penalties can be used to ensure that the estimated parameter values and derived quantities meet certain restrictions. For example, exploitation rate constraints on mortality events (i.e., fisheries) that are not violated (otherwise there is nothing to prevent the model from having abundances so low that the recorded catches could not have been taken); penalties on category transitions (to ensure there are enough individuals to move); penalties such that estimated values are similar or smooth, etc.

Equation 6.1 can be reduced to a penalised likelihood equation if all priors are assumed to be uniform. This is because uniform priors have no contribution to the objective function so Equation 6.1 reduces to the likelihood components plus penalties.

6.3 Specifying the parameters to be estimated

The parameters to be estimated (estimables) are defined using `@estimate` commands (see Section 10).

For example, a `@estimate` command block

```
@estimate male.m
parameter process[NaturalMortality].m{male}
lower_bound 0.1
upper_bound 0.4
type uniform
```

See Section 3.4.5 for information on how to specify the parameter name. At least one parameter is required to be estimated if doing an estimation `-e`, profile `-p`, or MCMC `-m` run. Initial values for the parameters to be estimated are required, and these values are used as the starting values for the minimiser. However, these values may be overwritten if a set of alternative starting values is provided (i.e., using `casal2 -i`, see Section 3.3).

All parameters are estimated within the specified bounds. For each parameter estimated, the lower and upper bounds and the prior (`type`) (Section 6.7) must be specified. The bounds and the prior should be chosen carefully as they affect the values over which the minimisers search. Some minimisers convert the lower and upper bounds 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 or fix the others by setting the the lower and upper bounds to the same value as the initial value.

6.4 Point estimation

Point estimation is invoked with `casal2 -e`, which attempts to find a minimum of the objective function. Casal2 has multiple minimisation algorithms. There are two automatic differentiation (AD) minimisers: ADOL-C, and BetaDiff (the minimiser used in CASAL). There are also three non-automatic differentiation minimisers: numerical differences, DeltaDiff, and the differential evolution minimiser (`de_solver`). Automatic differentiation minimisers are recommended for more complex models as they are on average much faster and tend to find a more robust minimum when exploring a complex objective surface.

An important input parameter for most minimisers is the `tolerance` parameter. This is the gradient of the objective function, and is used as the stopping rule to define the 'solution' (although a solution may be a local minimum and not the global minimum). Evaluating the robustness of a minimum can be tested with different starting values (i.e., using `-i free_parameter_file.txt`).

Start with the default `tolerance` parameter value of 1e-5 and decrease it while developing a model. For a given model, the parameter estimates when minimising with different tolerance may be different.

6.4.1 The numerical differences minimiser

See Section 10.2.5 for the command syntax.

The numerical differences minimiser uses a quasi-Newton minimiser which is a slightly modified implementation of the main algorithm of Dennis Jr and Schnabel (1996), and uses an *arcsin* transformation to ensure parameters remain within bounds.

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

- Successful convergence (suggests a local minimum has been found, at least).
- Convergence failure (a local minimum has not been found, although the results may be 'close enough').
- Convergence unclear (the minimiser halted but was unable to determine if convergence occurred. The result may be a local minimum, although this can be checked by restarting the minimiser at the final values of the estimated parameters).

The maximum number of quasi-Newton iterations and objective function evaluations allowed can be specified. If either limit is exceeded, the minimiser exits with a convergence failure. Set the maximum number of evaluations and iterations to values larger than the defaults of 300 and 1000, unless convergence is reached with fewer. An alternative starting point of the minimiser can be specified using `casal2 -i`.

The minimisers are local optimisation algorithms trying to solve a global optimisation problem. What this means is that, even if a 'successful convergence' is reached, the solution may be only a local minimum, and not a global one. To diagnose this problem, start multiple runs from different starting points and comparing the results, or do 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.

Note 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
```

6.4.2 The DeltaDiff numerical differences minimiser

See Section 10.2.4 for the command syntax.

DeltaDiff applies the same minimiser as Numerical Differences, expect that it uses *tan* rescaling for the parameters rather than *arcsin*. This minimiser may perform better than the Numerical Differences minimiser when parameters are very close to zero bounds.

6.4.3 The differential evolution minimiser

The differential evolution minimiser (`de_solver`) is a simple population-based, stochastic function minimizer, but is claimed to be quite powerful in solving minimisation problems. It is a method of mathematical optimization of multidimensional functions and belongs to the class of evolution strategy optimizers.

Initially, the procedure randomly generates and evaluates a number of solution vectors (the population size), each with p parameters. Then, for each generation (iteration), the algorithm creates a candidate solution for each existing solution by random mutation and uniform crossover. The random mutation generates a new solution by multiplying the difference between two randomly selected solution vectors by some scale factor, then adding the result to a third vector. Then an element-wise crossover takes place with probability P_{cr}, to generate a potential candidate solution. If this is better than the initial solution vector, it replaces it, otherwise the original solution is retained. The algorithm terminates 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 the model, this value can be set to a lower value and still find a robust solution.

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. Some results suggest 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 automatic differentiation minimisers or other more sophisticated algorithms have not been made.

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

6.4.4 The BetaDiff minimiser

An automatic differentiation 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
```

6.4.5 The ADOL-C minimiser

An automatic differentiation minimiser for non-linear models. See `https://projects.coin-or.org/ADOL-C` for more information. Users do have an option of defining what transformation to apply to convert the parameter $\theta \in [\theta_{LB}, \theta_{UB}]$ to $X \in [-1, 1]$, for which optimisation is done. The options are *sin* or *tan*. Initial model runs suggest this assumption will make a difference to convergence, particularly if there are poorly identified parameters which fall at the bounds, we have found the sin transform is more consistent with the betadiff minimiser. The sin transform

$$X = \frac{asin(2 * (\theta - \theta_{LB})/(\theta_{UB} - \theta_{LB}) - 1)}{1.57079633} \tag{6.2}$$

the consequence of this transformation is when X is back transformed to θ there is a penalty which is added to the minimisation to dissuade parameter values close to the bounds. This penalty is hidden from the reported objective function. If you are interested in it, you can run with `--loglevel medium` and it should be reported. The back transformation follows,

$$\theta = \theta_{LB} + (\theta_{UB} - \theta_{LB}) * (sin(X * 1.57079633) + 1)/2; \tag{6.3}$$

and penalty

$$if(-0.9999 - X < 0) \qquad penalty+ = (X + 0.9999)^2 \tag{6.4}$$

$$if(X - 0.9999 < 0) \qquad penalty+ = (X - 0.9999)^2 \tag{6.5}$$

The Tan transform uses transformation

$$X = tan(((\theta - \theta_{LB})/(\theta_{UB} - \theta_{LB}) - 0.5) * \pi) \tag{6.6}$$

and back transform

$$\theta = ((atan(X)/\pi) + 0.5) * (\theta_{UB} - \theta_{LB}) + \theta_{LB} \tag{6.7}$$

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

6.5 Posterior profiles

If profiles are run using the command `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 the 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. The initial point estimate should be compared with the profile results, to check at least that none of the other points along the profile have a better objective function value than the initial 'minimum'.

The parameters to be profiled are specified, and optionally the number of steps, and lower bound and upper bound, for each parameter. In the case of vector parameters, the element(s) of the vector to be profiled are specified.

The initial starting point for the estimation can also be specified using `casal2 -i file`, which may improve the minimiser performance for the profiles.

If the profile results are not reasonable, 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 may also be useful to try other minimisers and compare the results. An example excerpt follows, but also see the syntax at Section 10.4.

```
@profile B0
parameter process[Recruitment_east].b0
steps 10
lower_bound 10000
upper_bound 100000
## you can force other parameters to be the same
same process[Recruitment_west].b0
```

To run a Casal2 you will need to supply the following reports

```
@report profile
type profile

@report estimate_values
type estimate_value

@report objective_scores
type objective_function
```

6.6 Bayesian estimation

Casal2 can use Markov chain Monte Carlo (MCMC) functionality to generate a sample from the posterior distribution of the estimated parameters with command `casal2 -m` or `casal2 -M MPD_file` and output the sampled values, optionally keeping only every nth set of values.

As Casal2 has no post-processing capabilities. Casal2 cannot produce MCMC convergence diagnostics. To calculate these diagnostics, use a package such as BOA, plot/summarize the posterior distributions of the output quantities, and/or use a general-purpose statistical package such as **R**.

Bayesian methodology and MCMC are both large and complex topics. 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 10.3 for the Casal2 commands used in an MCMC Bayesian analysis.

Casal2 implements two methods for MCMC. The first is a straightforward implementation of the random walk 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 (the burn-in period) and every Nth element of the remainder is taken as the posterior sample. The second is Hamiltonian Monte Carlo. This uses similar subcommands as the random walk Metropolis-Hastings algorithm. In both cases, the chain is produced by taking an initial point x_0 and repeatedly applying the following rule, where x_i is the current point:

1. Draw a candidate step s from a proposal distribution J, which should be symmetric i.e., $J(-s) = J(s)$
2. Calculate $r = min(\pi(x_i + s)/\pi(x_i), 1)$
3. 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 (although can alternatively be supplied from a previous estimation run), 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 starting point of the point estimate minimiser can be specified using the command `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 determine a reasonable approximation to the Hessian.

There are currently three options for the starting point of the MCMC. It can be from either the MPD, an estimate supplied in a `-i` file, or from a random 'jump' from the point estimate. In the later case, it is generated from a multivariate normal distribution, centred on the start point, with covariance equal to the inverse Hessian multiplied by a user-specified constant (using the subcommand `start`). This may be useful if the chain gets 'stuck' at the point estimate, or if you wish to generate multiple chains from for later MCMC diagnostic tests.

Note that if a number of parameters are at bounds at the start point, then a random jump from this point may fail to find a suitable candidate that is within bounds. A total of 10 000 attempts are made before the algorithm exits with an error. The subcommand `adjust_parameters_at_bounds` can be used to identify these parameters and set their start point at a random uniformly generated point between the lower and upper bounds.

The chain moves in natural space, i.e., no transformations are applied to the estimated parameters. The default proposal distribution is a either a multivariate normal or a multivariate Student's t distribution centred on the current point, with covariance matrix equal to a matrix based on the approximate covariance produced by the minimiser, multiplied by the step-size factor.

The following steps define how the initial covariance matrix of the proposal distribution is calculated:

1. The covariance matrix is taken as the inverse of the approximate Hessian from the quasi-Newton minimiser.
2. 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.
3. 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` multiplied by the difference between the parameters' lower and upper bound, then the variance is changed, without changing the associated correlations, to $k = \text{min\_diff}(upper\_bound_i - lower\_bound_i)$. This is done by setting

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

    for $i \neq j$, and $\text{var}(i)' = k$
    - `@mcmc.adjustment_method=correlation`: that if the variance of the ith parameter is non-zero and less than `@mcmc.min_difference` multiplied by the difference between the parameters' lower and upper bounds, 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.

4. The `@mcmc.step_size` (a scalar factor applied to the covariance matrix to improve the acceptance probability) is set 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).

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:

- The step-size changes adaptively at one or more sample numbers (See next paragraph for details on the step-size adaptation methods)
- The entire covariance matrix changes adaptively at one or more sample numbers. At each adaptation, the covariance matrix is replaced with an empirical covariance matrix, derived from the MCMC chain. The idea is that an empirical covariance is a better approximation of the proposal distribution than the inverse of the Hessian matrix, and can improve convergence and mixing of the chain.

The two options to adapt the step-size are `double_half` or `ratio`, defined by the subcommand `adapt_stepsize_method`. The `double_half` method is the same as the method used in CASAL for adapting the step-size (see Gelman et al. (1995) for justification). The algorithm for `double_half` is, at each adaptation, the step-size is doubled if the acceptance rate since the last adaptation is more than 0.5, or halved if the acceptance rate is less than 0.2. The `ratio` method adapts the current step-size by the acceptance rate since the last adaptation multiplied by 4.1667 so-as to approach an acceptance rate of ≈ 0.24. See Sherlock and Roberts (2009) for justification on the choice of that acceptance rate.

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

All modified versions of the covariance matrix are printed to the standard output, but only the initial covariance matrix (inverse Hessian) is saved to the objectives file (see Section 12.1.15).

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 that, to start, 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 had at least this number of transitions, then it is also a fatal error. If this test is passed, the set of points is systematically sub-sampled down to 1000 points (and it must be at least this long to start with).

The probability of acceptance for each jump is 0 if the jump would move a parameter value outside of its bounds, 1 if it improves the posterior, or $(newposterior/oldposterior)$ otherwise. How often the position of the chain is recorded is specified with the `keep` parameter. For example, with `keep 10`, only every 10th sample is recorded.

The option to specify that some of the estimated parameters are fixed during the MCMC is available via the `mcmc_fixed` in the `@estimate` block. If the chain starts at the point estimate or at a random location, these fixed parameters are set to their values at the end of estimation during the MCMC phase.

If the start of the chain is specified with the command `casal2 -i`, these fixed parameters are set to the values in the file.

Restarting an MCMC chain: in the case where an MCMC chain was halted or interrupted, the MCMC chain can be restarted from where it finished with

```
casal2 -R MPD_file --objective-file objectives_file --sample-file samples_file
```

where `Objective_file_name` is the file name for the objective function report and `Sample_file_name` is the file name for the sample report from a MCMC chain.

The posterior sample can be used for (projections (Section 5.12)) or simulations (see Section 7.5) with the values supplied with the command `casal2 -i file --t`.

A multivariate Student's t distribution can be used as an alternative to the multivariate normal proposal distribution. If you request multivariate Student's t proposals, you can change the degrees of freedom from the default of 4. As the degrees of freedom decreases, the t distribution becomes more heavy tailed. This may lead to better convergence properties. Note the default is the multivariate Student's t.

Given a posterior (sub)sample, Casal2 can calculate a list of output quantities for each sample point (see Section 8 specifically tabular report). These quantities can be output to a file (with the command `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 5.12). The advantage of this is that the parameter uncertainty, as expressed in the posterior distribution, can be included into the risk estimates.

Casal2 will error out if asked to run MCMC for a model that does not contain the following reports,

```
@report mcmc_samples
type mcmc_sample

@report mcmc_objectives
type mcmc_objective
```

The default file name for these reports are `samples` and `objectives`. The `write_mode` will default to increment suffix, which means each time you re-run an MCMC in a directory with the same file name, it will increment the extension.

6.7 Priors

In a Bayesian analysis, a prior is required for every parameter that is being estimated. There are no default priors.

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 the 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):

6.7.1 Uniform

$$-\log(\pi(p)) = 0 \tag{6.8}$$

6.7.2 Uniform-log

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

$$-\log(\pi(p)) = \log(p) \tag{6.9}$$

6.7.3 Normal

The normal distribution with mean μ and standard deviation with c.v c

$$-\log(\pi(p)) = 0.5\left(\frac{p-\mu}{c\mu}\right)^2 \tag{6.10}$$

6.7.4 Normal with standard deviation

The normal distribution with mean μ and standard deviation σ

$$-\log(\pi(p)) = 0.5\left(\frac{p-\mu}{\sigma}\right)^2 \tag{6.11}$$

6.7.5 Lognormal

The lognormal distribution with mean μ and c.v. c

$$-\log(\pi(p)) = \log(p) + 0.5\left(\frac{\log(p/\mu)}{s} + \frac{s}{2}\right)^2 \tag{6.12}$$

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

6.7.6 Normal-log

The normal-log distribution with mean μ and c.v. c

Similar to the lognormal prior, but with the mean (*mu*) and standard deviation (*sigma*) specified in log space.

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

6.7.7 Beta

The Beta distribution with mean μ and standard deviation σ, and range parameters A and B

$$-\log(\pi(p)) = (1-m)\log(p-A) + (1-n)\log(B-p) \tag{6.13}$$

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 $\tau \leq 0$.

6.7.8 Student's t

The Student's t distribution with location (mean) μ, scale σ, and degrees of freedom ν where the pdf is defined as

$$\begin{aligned} f(x|\mu,\sigma,\nu) &= \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)}\,\frac{1}{\sqrt{\nu\pi}\,\sigma}\left(1+\frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-(\nu+1)/2} \text{ and} \\ -log(\pi(p)) &= -\log(f(x|\mu,\sigma,\nu)) \end{aligned} \tag{6.14}$$

The Student's t prior is Cauchy with $\nu = 1$ and is equivalent to the normal as $\nu \to$ inf.

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:

```
## 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
```

6.8 Penalties

Penalties are associated with processes and can be used to enforce parameter value or derived quantity restrictions or model outputs that are invalid by adding a penalty to the objective function. For example, estimated parameter values can be restricted so that a known mortality event removes enough individuals from the population 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. Many of the penalties that were available in CASAL are implemented as additional priors in Casal2(see Section 6.9).

For penalties, a multiplier is required to be specified, and the objective function is increased by this multiplier multiplied by the penalty value. In some cases the multiplier may need to be quite large to prohibit some model behaviour.

Penalties are implemented 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`

For these processes, two types of penalties can be defined: on the natural scale (the default) and on the log scale. Both of these types add a penalty value of the squared difference between the observed value (e.g., 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), multiplied by the penalty multiplier.

The natural scale penalty calculates the squared difference on a natural scale, and the log scale penalty calculates the squared difference of the logged values.

For example:

```
@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;

$$Penalty = (X_1 - X_2)^2 \tag{6.15}$$

or if `log_scale true`

$$Penalty = (log(X_1) - log(X_2))^2 \tag{6.16}$$

where, for example, X_1 is observed catch biomass and X_2 is the estimated catch biomass. Penalties are usually applied in situations when numbers or weight are known. Another example is for tagging, where the number of individuals that were tagged in a given year is known, so a penalty can be used to restrict the model to estimate reasonable values for the numbers of tagged individuals in that year.

6.9 Additional priors

Additional priors are optional additional priors or penalties that can be applied to encourage vectors to be smooth or have some average, or apply a prior to parameter or group of parameters.

The types of additional priors available include smoothing and averaging of vector parameters, a uniform log or lognormal prior on a single parameter, and a prior on the sum of a list of parameters. The additional priors are described below.

6.9.1 Vector smoothing

The `vector_smoothing` additional prior is applied to a vector parameter. Sum of squares of r^{th} differences, optionally on a log scale. This encourages the vector to be like a polynomial of degree $(r-1)$. A range of the vector to be "smoothed" can be specified (and if not, the smoother is applied to the entire vector). However, this restriction must be specified by an index of the vector and must be between 1 and the length of the vector, inclusive.

6.9.2 Vector average

The `vector_average` additional prior is applied to a vector parameter and restricts the vector to average arithmetically with `method=k` or `method=m`, or geometrically to $exp(k)$ when `method=m`. Typically used with $k = 1$ with `method=k` or `method=l`, or $k = 0$ with `method=m`, to restrict the recruitment_multipliers to centre on 1. Optionally, indices can be chosen or excluded outside a given set of bounds.

Methods available for the type `vector_average` are `l`, `k`, `m`. For a target vector parameter $\mathbf{X}$ and target mean k, the contribution to the objective score is

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

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

6.9.3 Lognormal

Apply a `lognormal` additional prior with mean μ and c.v. c to a single parameter,

$$-\log(\pi(p)) = \log(p) + 0.5\left(\frac{\log(p/\mu)}{s} + \frac{s}{2}\right)^2 \tag{6.17}$$

6.9.4 Uniform

Apply a `uniform` additional prior to a single parameter,

$$-\log(\pi(p)) = 0 \tag{6.18}$$

6.9.5 Uniform-log

Apply a `uniform_log` additional prior to a single parameter,

$$-\log(\pi(p)) = \log(p) \tag{6.19}$$

6.9.6 Element difference

Apply an `element_difference` additional prior to encourage two vectors of equal length to be similar, where the value is the difference between the elements of each vector, such that,

$$-\log(\pi(p_1, p_2)) = \sum_{i=1}^{n} (p_{1,i} - p_{2,i})^2 \tag{6.20}$$

6.9.7 Beta

Apply a `beta` additional prior to a single parameter with mean μ and standard deviation σ, and range parameters A and B, for parameter value = p

$$-\log(\pi(p)) = (1-m)\log(p-A) + (1-n)\log(B-p) \tag{6.21}$$

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 $\tau \leq 0$.

6.9.8 Sum

Apply a `sum` additional prior on the sum of a list of parameters, using either a normal or lognormal distribution.

The normal distribution has mean μ and standard deviation with c.v c

$$-\log(\pi(p)) = 0.5\left(\frac{p-\mu}{c\mu}\right)^2 \tag{6.22}$$

And the lognormal distribution has mean μ and c.v. c

$$-\log(\pi(p)) = \log(p) + 0.5\left(\frac{\log(p/\mu)}{s} + \frac{s}{2}\right)^2 \tag{6.23}$$

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

This additional prior can be used to encourage the sum of a list of parameters to have value *mu*.

6.9.9 Applying additional priors

All parameters that can be estimated can also have an additional prior. For parameters that are not estimated within a specific model run, additional priors can be applied to.

- `selectivity[Selectivity_label].values{i:j}`.

  This subcommand applies a selectivity to the value by age (for ages *i* through *j*). This option is available only for certain types of selectivities (`all_values`, `all_values_bounded`, `double_exponential`).
- `catchability[Catchability_label].q`

  This subcommand is for catchabilities that are of type `nuisance` only. Since `nuisance` *q*s are not free parameters, additional priors can be applied to replicate CASAL models with `@estimate` blocks in nuisance *q*s.

6.10 Parameter transformations

Casal2 has multiple methods to transform a parameter into a different "space". Transformations are implemented to try and achieve "better" model optimisation. Complex population models can have highly correlated parameters so transforming them is a method of addressing confounded parameters, and "help" the minimisers find a "global" solution faster. To read more about transformations and get a better understanding of why they are used, see Gilks et al. (1995), specifically chapter 6.

To transform a parameter the `@parameter_transformation` block is used. For example if users wanted to estimate $\log R_0$ instead of R_0, they could do the following,

```
## define transformation
@parameter_transformation log_R0
type log
parameters process[Recruitment].r0

## define @estimate for the log parameter
@estimate log_R0
type uniform
parameter parameter_transformation[log_r0].log_parameter
lower_bound 1
upper_bound 25
```

The available parameter transformations are,

1. Log (Univariate transformation) Section 6.10.2 - 1
2. Inverse (Univariate transformation) Section 6.10.2 - 2
3. Difference (Bivariate transformation) Section 6.10.2 - 3
4. Average difference (Bivariate transformation) Section 6.10.2 - 4
5. Log sum (Bivariate transformation) Section 6.10.2 - 5
6. Orthogonal (Bivariate transformation) Section 6.10.2 - 6
7. Logistic (Univariate transformation) Section 6.10.2 - 7
8. Sum to one (Bivariate transformation) Section 6.10.2 - 8
9. Simplex (Multivariate transformation) Section 6.10.2 - 9
10. Square root (Univariate transformation) Section 6.10.2 - 10

To see the parameters that can be used in `@estimate` block for each estimable transformation see the `estimable parameter` description in Section 6.10.2.

When users estimate a transformed parameter they have the option of defining the prior for the transformed parameter or for the parameter in natural space. An example of when the later has been used. Say a meta-analysis has been done on the catchability parameter, for which an *a priori* assumption can be made, but the user wants to estimate log transformed catchability for optimisation reasons. In this instance users are required to use the subcommand `prior_applies_to_restored_parameters`. If this is true the prior will be applied to the untransformed parameter and a Jacobian will be added (if it is known) to account for the change in variable. If the Jacobian is false then the prior refers to the transformed parameter and no adjustments are needed. If users specify to calculate a Jacobian and the estimate is not a `parameter_transformation` Casal2 will print a warning and ignore this input.

6.10.1 Transform with Jacobian

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

$$supp(X) = \{x|p_X(x) > 0\} \tag{6.24}$$

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 resulting probability density function of Y.

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

$$p_Y(y) = p_X(f^{-1}(y)) \left| \frac{\partial}{\partial y} f^{-1}(y) \right| \tag{6.25}$$

where $\left| \frac{\partial}{\partial y} f^{-1}(y) \right|$ si the Jacobian adjustment is the absolute derivative of the transform. 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.

In equation 6.25 the term $p_X(f^{-1}(y)) = p_X(X)$ and in a Bayesian context is the prior of the untransformed variable/parameter. Casal2 defines the objective function as the negative log-likelihood. This means $\left| \frac{\partial}{\partial y} f^{-1}(y) \right|$ needs to be times by a negative log, as it is currently defined as an adjustment to the density.

6.10.2 Transformation types

1. `type log` : natural logarithm transformation
   `Jacobian defined = true`
   `estimable parameter = log_parameter`
   $Y = log(X)$
   $f() = log()$
   $f^{-1}() = exp()$

   $$log \left| \frac{\partial}{\partial y} exp(y) \right| = log \left| exp(y) \right| = log(x)$$

```
@parameter_transformation log_R0
type log
parameters process[Recruitment].r0

@estimate log_R0
```

```
type uniform
parameter parameter_transformation[log_r0].log_parameter
lower_bound 1
upper_bound 25
```

2. `inverse`
   `Jacobian defined = true`
   `estimable parameter = inverse_parameter`
   $Y = X^{-1}$

   $$log\left|\frac{\partial}{\partial y}\frac{1}{y}\right| = log\left|y^{-2}\right| = -2log(y)$$

```
@parameter_transformation inverse_R0
type inverse
parameters process[Recruitment].r0

@estimate inverse_R0
type uniform
parameter parameter_transformation[inverse_R0].inverse_parameter
lower_bound 0.001
upper_bound 1
```

3. `difference` : two parameters X_1 and X_2 are transformed to X_1 and $X_1 - d$, where d is the difference between the original parameters.
   `Jacobian defined = true`
   `estimable parameter = difference_parameter`
   $Y_1 = X_1$
   $Y_2 = X_1 - d$
   Restore transformations
   $X_1 = Y_1$
   $X_2 = X_1 - d$

```
@parameter_transformation diff
type difference
parameters process[InstantMortality].m{male} process[InstantMortality].m{female}
difference_parameter 0.05

@estimate diff_m
type uniform
parameter parameter_transformation[diff].difference_parameter
lower_bound -0.5
upper_bound 0.5
```

4. `average_difference` : two parameters X_1 and X_2 are transformed 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.
   `Jacobian defined = false`
   `estimable parameter = average_parameter, difference_parameter`
   $Y_1 = \frac{X_1+X_2}{2}$
   $Y_2 = (Y_1 - X_2)^2$
   Restore transformations
   $X_1 = Y_1 + 0.5Y_2$
   $X_2 = X_1 - 0.5Y_2$
   $\left|\frac{\partial}{\partial y} f^{-1}(y)\right|$ Hasn't been assessed (i.e it could exist)

```
@parameter_transformation avg_diff
type average_difference
parameters process[InstantMortality].m{male} process[InstantMortality].m{female}
```

```
@estimate avg_m
type uniform
parameter parameter_transformation[avg_diff].average_parameter
lower_bound 0.01
upper_bound 1

@estimate diff_m
type uniform
parameter parameter_transformation[avg_diff].difference_parameter
lower_bound -0.5
upper_bound 0.5
```

5. `log_sum` : two parameters X_1 and X_2 are transformed to Y_1 and Y_2, where Y_1 is the natural logarithm of the sum of X_1 and X_2. Y_2 describes the proportion of the sum with respect to X_1
   `Jacobian defined = false`
   `estimable parameter = log_total_parameter, total_proportion_parameter`
   $Y_1 = ln(X_1 + X_2)$
   $Y_2 = X_1/(X_1 + X_2)$
   Restore transformations
   $X_1 = exp(Y_1)Y_2$
   $X_2 = exp(Y_1)(1 - Y_2)$
   $\left|\frac{\partial}{\partial y} f^{-1}(y)\right|$ Hasn't been assessed (i.e it could exist)

```
@parameter_transformation log_total_r0
type log_sum
parameters process[Recruitment_east].r0 process[Recruitment_west].r0

@estimate log_total_r0
type uniform
parameter parameter_transformation[log_total_r0].log_total_parameter
lower_bound 4
upper_bound 25

@estimate prop_r0_east
type uniform
parameter parameter_transformation[log_total_r0].total_proportion_parameter
lower_bound 0.001
upper_bound 0.8
```

6. `orthogonal` : two parameters X_1 and X_2 are transformed to Y_1 and Y_2, where Y_1 is the multiplication of X_1 and X_2. Y_2 is the division of X_1 and X_2
   `Jacobian defined = true`
   `estimable parameter = product_parameter, quotient_parameter`
   $Y_1 = X_1 X_2$
   $Y_2 = X_1/X_2$
   Restore transformations
   $X_1 = \sqrt{Y_1 Y_2}$
   $X_2 = \sqrt{Y_1/Y_2}$
   $\left|\frac{\partial}{\partial y} f^{-1}(y)\right| = 2Y_2$

```
@parameter_transformation orthogonal_trans
type orthogonal
parameters process[Recruitment].r0 catchability[CPUEQ].q
```

```
@estimate B0_times_q
type uniform
parameter parameter_transformation[orthogonal_trans].product_parameter
lower_bound 0.1
upper_bound 2500

@estimate B0_divide_q
type uniform
parameter parameter_transformation[orthogonal_trans].quotient_parameter
lower_bound 0.001
upper_bound 1e8
```

7. `type logistic` : logistic transformation
   `Jacobian defined = true`
   `estimable parameter = logistic_parameter`
   $Y = logit(\frac{X-lb}{ub-lb})$
   $f^{-1}() = lb + (ub-lb)logit^{-1}()$

   $$\left|\frac{\partial}{\partial y} lb + (ub-lb)logit^{-1}(y)\right| = (ub-lb)logit^{-1}(y)\left(1-logit^{-1}(y)\right)$$

```
@parameter_transformation logistic_R0
type logistic
parameters process[Recruitment].r0
lower_bound 10000
upper_bound 600000

@estimate logistic_R0
type uniform
parameter parameter_transformation[logistic_R0].logistic_parameter
lower_bound -1000 # theoretically -Inf
upper_bound 1000 # theoretically Inf
```

8. `SumToOne` : given two parameters X_1 and X_2 that have the constraint $\sum_{i=1}^{2} X_i$, estimate X_1 only given $X_2 = 1 - X_1$
   `Jacobian defined = false`

```
@parameter_transformation total_r0
type sum_to_one
parameters process[Recruitment_east].r0 process[Recruitment_west].r0

@estimate total_r0
type uniform
parameter parameter_transformation[log_total_r0].total_parameter
lower_bound 4
upper_bound 25

@estimate prop_r0_east
type uniform
parameter parameter_transformation[log_total_r0].proportion_parameter
lower_bound 0.001
upper_bound 0.8
```

9. `simplex` : given the vector of parameters $\mathbf{X} = (X_1, \ldots, X_n)$ which either has the constraint $\sum_{i=1}^{n} X_i = 1$ or $\sum_{i=1}^{n} X_i = n$. Then the simplex is a suitable transformation. It translates to a new vector parameter $\mathbf{Y} = (Y_1, \ldots, Y_{n-1})$ which has unconstrained parameter space i.e $Y_i \in (-\infty, \infty)$. Note that the calculation of the Jacobian for the simplex is still experimental and may not be suitable in all circumstances.

This transformation follows the implementation in stan, where an intermediate variable Z_i is used. The transformation going from **X** to **Y** follows

$$Z_i = \frac{X_i}{1 - \sum_{j=1}^{i-1} X_j}$$

and

$$Y_i = logit(Z_i) - log\left(\frac{1}{n-i}\right)$$

The inverse transformation going from **Y** to **X** follows

$$Z_i = logit^{-1}\left(Y_i + log\left(\frac{1}{n-i}\right)\right)$$

and

$$X_i = \left(\sum_{j=1}^{i-1} X_j\right) Z_i \qquad \text{for } i < n$$

$$X_n = 1 - \sum_{i=1}^{n-1} X_i \qquad \text{for } i = n$$

The Jacobian for the density is evaluated as follows,

$$|det\ J| = \prod_{i=1}^{n-1} Z_i (1 - Z_i) \left(1 - \sum_{j=1}^{i-1} X_j\right)$$

```
Jacobian defined = true
estimable parameter = simplex


@parameter_transformation simplex_ycs
type simplex
sum_to_one false
parameters process[Recruitment].ycs_values{1950:2018}
prior_applies_to_restored_parameters true

@estimate simplex_ycs
type uniform
parameter parameter_transformation[simplex_ycs].simplex
lower_bound -10
upper_bound 10
```

10. `type sqrt` : square root transformation

```
Jacobian defined = true
estimable parameter = sqrt_parameter
```

$Y = sqrt(X)$
$f() = sqrt()$
$f^{-1}(x) = x * x$

$$log\left|\frac{\partial}{\partial y}(y^2)\right| = sqrt\left|(y^2)\right| = sqrt(x)$$

```
@parameter_transformation sqrt_R0
type sqrt
parameters process[Recruitment].r0
```

```
@estimate sqrt_R0
type uniform
parameter parameter_transformation[sqrt_r0].sqrt_parameter
lower_bound 1
upper_bound 25
```

If users want to force other parameters in the system to be the same as an estimated transformation, this can be done by creating multiple `@parameter_transformation` blocks. For example if there were multiple categories (spawning and non spawning males and females) and the average difference parametrisation was used to estimate natural mortality. The non-spawning components can be set the same as the spawning values using the following syntax.

```
@categories
format sex.maturity
names male.spawn female.spawn male.nonspawn female.nonspawn

@parameter_transformation avg_diff_spawn
type average_difference
parameters process[InstantMortality].m{male.spawn}
           process[InstantMortality].m{female.spawn}

@parameter_transformation avg_diff_non_spawn
type average_difference
parameters process[InstantMortality].m{male.nonspawn}
           process[InstantMortality].m{female.nonspawn}

@estimate avg_m
type uniform
parameter parameter_transformation[avg_diff_spawn].average_parameter
same parameter_transformation[avg_diff_non_spawn].average_parameter
lower_bound 0.01
upper_bound 1

@estimate diff_m
type uniform
parameter parameter_transformation[avg_diff_spawn].difference_parameter
same parameter_transformation[avg_diff_non_spawn].average_parameter
lower_bound -0.5
upper_bound 0.5
```

This can be done for any set of parameters in the system, for example if you had multiple recruitment dynamics and wanted to estimate a joint steepness parameter with the log transformation, you would need to create multiple blocks and force them in the same.

7 The observation section: observations and their likelihoods

The command and subcommand syntax for the observation section is given in Section 11.1.

7.1 Observations

The objective function calculates the goodness-of-fit of the model to the observation data. Observations are typically supplied at an instance in time, over a group of aggregated categories. Most observations are sampled over 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-length.

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 type is observations that are associated with a process, and the second are associated with a mortality block (See Section 5.3.2.

Observations for a process are indicated by their type — these use the word process as a part of the type name, e,.g., `@observation.type=abundance` is an observation of relative abundance that occurs during a mortality block within a time step, and `@observation.type=process_abundance` is a observation of relative abundance that occurs during a process within a time step.

A robustification constant is used in the likelihood (`delta`) to avoid divide by zero errors. This defaults to 1e-11.

7.1.1 Mortality block associated observations

All observations within this class are calculated similarly. That is, the expected values are calculated at the beginning of the mortality block and at the end of the mortality block. Casal2 then uses a linear interpolation to approximate the expected values part way through a mortality block using the subcommand `time_step_proportion`. This feature could be useful if a survey occurs part way through an exploitation phase, which may be part way through a fishing season when modelling a fish population. Each observation in this class will evaluate different expectations of the partition (explained in the following descriptions).

The observation `types` available with this class of observations are:

- `abundance`
- `biomass`
- `proportions_at_length`
- `proportions_by_category`
- `tag_recapture_by_length`

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 length and category) while an abundance observation is the number of individuals.

Each observation is for a given year and time-step, for some selected length classes of the population (for a range of length classes multiplied by a selectivity), for aggregated categories. Furthermore, the label of the catchability coefficient q is required; q can either be estimated of fixed. For absolute abundance or absolute biomass observations, define a catchability where $q = 1$. Catchabilities can be estimated as either free parameters or as nuisance parameters (see Section 7.4).

The observations can be supplied for any set of categories. For example, for a model with the two categories *male* and *female*, an observation of the total abundance/biomass (male + female) or male-only abundance/biomass could be provided. 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 requires that an observation is 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 length classes (via the length range and selectivity) and categories at both the beginning and end of a mortality block. Casal2 will approximate the expectation part way through the mortality block using the `time_step_proportion`. The default `time_step_proportion` value is 0.5. Casal2 does linear interpolation between the start and end abundance (or biomass) from the mortality block.

For an abundance observation the expected value is

$$E_{i,1} = \sum_{c=1}^{} \sum_{a=1}^{A} S_{a,c} N_{a,c,i,1} \tag{7.1}$$

$$E_{i,2} = \sum_{c=1}^{} \sum_{a=1}^{A} S_{a} N_{a,c,i,2} \tag{7.2}$$

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 length 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, then $N_{a,c,i,1}$ in Equations (7.1) and (7.2) is replaced with $N_{a,c,i,1}\bar{w}_{a,c}$, where $\bar{w}_{a,c}$ is the mean weight of category c at length 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), the linear interpolation is

$$E_i = |E_{i,1} - E_{i,2}| M_i \tag{7.3}$$

For each year of observations, the observation table `table obs` has a row with year in the first column, the observation per category in the middle column(s), and the error value in the final column:

```
@observation MyAbundance
type abundance
years 1999
...
categories male
table obs
1999 1000 0.10
end_table
...
```

For an observation aggregated over multiple categories:

```
@observation MyAbundance
type abundance
```

```
years 1990 1991
...
categories male+female
table obs
1990 1000 0.10
1991 1200 0.12
end_table
...
```

For observations for multiple categories:

```
@observation MyAbundance
type abundance
years 1990 1991
...
categories male female
table obs
1990 550 450 0.10
1991 700 500 0.12
end_table
...
```

To define a biomass observation instead of an abundance observation, use

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

Proportions-by-category observations

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

The observation is supplied for a given year and time-step, for selected length classes of the population (i.e., for a range of lengths multiplied by a selectivity).

The length range must be length bins defined in the partition; the upper end of the 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*, observations of the proportions of males in the population at each length class might be provided. 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.

For example,

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

defines the proportion of males in each length class as a proportion of males + females. Casal2 then requires that there will be a vector of proportions supplied, with one proportion for each length class within the defined

range, i.e., if the age range was 3 to 10, then 8 proportions should be supplied (one proportion for each of the classes 3, 4, 5, 6, 7, 8, 9, and 10). The expected values will be the expected ratios of male to male + female within each of these age classes, after applying the selectivities at the year and time-step specified.

Casal2 calculates the expected values by summing over the lengths (via the range and selectivity)

For example,

```
@observation MyProportions
type proportions_by_category
years 1990 1991
...
categories male
categories2 female
length_bins 1 2 3 4 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
...
```

Proportions-at-Length

Proportions-at-length observations are observations of the relative number of individuals by length, via some selectivity.

The observation is supplied for a given year and time-step, for some selected length bins of the population (i.e., for a range of lengths multiplied by a selectivity). Note that the categories defined in the observations must have an associated selectivity, defined by `selectivities`.

The length bins supplied must be a subset of the length bins defined in the `@model.b`lock.

Proportions-at-length observations can be supplied as

- a set of proportions for a single category,
- a set of proportions for multiple categories, or
- a set of proportions across aggregated categories.

The method of evaluating expectations are the same for all three types of proportions. The definitions of these proportions and the expected dimensions of observation and error inputs that Casal2 expects for each respective proportion type are described below with examples.

Like all types of observations that are associated with the mortality block, Casal2 will evaluate the numbers at length before and after the mortality block for the specified time step of the observation, and applying the user-defined selectivity. Casal2 then generates the 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.

Defining an observation for a single category is used to model a set of proportions of a single category by length class. For example, to specify that the observations are of the proportions of male within each length class, then the subcommand `categories` for the `@observation[label].type=proportion_by_length` command is

```
categories male
```

Casal2 then requires that there will be a single vector of proportions supplied, with one proportion for each length class within the defined length range.

```
@model
```

```
length_bins 6 8 10 12 14 16 18 20 22 24 26
@observation MyProportions
type proportions_at_length
...
categories male
length_bins 10 12 14 16 18 20
years 1990
table obs
1990 0.01 0.09 0.20 0.20 0.35
end_table
...
```

Defining an observation for multiple categories extends the single category observation definition. It is used to model a set of proportions over several categories by length class. For example, to specify that the observations are of the proportions of male or females within each length class, then the subcommand `categories` for the `@observation[label].type=proportion_by_length` command is

```
categories male female
```

Casal2 then requires that there will be a single vector of proportions supplied, with one proportion for each category and length class combination, and that these proportions sum to one across all lengths and categories.

For example,

```
@observation MyProportions
type proportions_at_length
...
categories male female
length_bins 2 4 6 8 10
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 length 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 length class, then the subcommand `categories` for the `@observation[label].type=proportion_by_length` command is

```
categories male + female
```

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

For example,

```
@observation MyProportions
type proportions_at_length
...
years 1990 1991
categories male+female
```

```
length_bins 2 4 6 8 10
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 latter 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 lengths 2 through 8, a total of 8 proportions are required for each year

```
@observation MyProportions
type proportions_at_length
...
categories male_immature+male_mature female_immature+female_mature
length_bins 2 4 6 8
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-category-by-length observations

Proportions-category observations are observations of relative number of individuals between categories within a range of length classes.

The observation is supplied for a given year and time-step, for selected length classes of the population.

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 two categories *male* and *female*, observations of the proportions of males in the population at each length class might be provided. The subcommand `categories` defines the categories for the numerator in the calculation of the proportion, and the subcommand `total_categories` supplies the categories to be used in the denominator of the calculation of which the the values of `categories` must be included. In addition, each category must have an associated selectivity, defined by `selectivities` for the numerator categories and `total_selectivities` for the additional categories used in the denominator.

For example,

```
categories male
total_categories male
selectivities maturity-selectivity
total_selectivities [type=constant; c=1] ## total population
```

defines the proportion of mature males in each length class as a proportion of all males. Another example may be males to females

```
categories male
total_categories male+female
selectivities [type=constant; c=1]
total_selectivities [type=constant; c=1] ## total population
```

Casal2 then requires that there will be a vector of proportions supplied, with one proportion for each length class within the defined length bin range. The expected values will be the expected proportion of `categories` to `total_categories`, after applying the selectivities at the year and time-step specified.

For example,

```
@observation mature_males
type proportions_by_category
years 1990 1991
categories male
total_categories male
selectivities maturity-selectivity
total_selectivities [type=constant; c=1] ## total population
length_bins 2 4 6 8 10
likelihood binomial
table obs
1990 0.01 0.05 0.6 1.00 0.820
1991 0.02 0.06 0.70 0.91 0.988
end_table
...
```

Tag recaptures 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, a tag-release process (Section 5.3.4) needs to be defined. 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 number can be specified by categories, or for combinations of categories. The precise content of the scanned and recaptured observations depends on the sampling method.

The options for tag-recaptures are available:

- size: both the scanned and recaptured are vectors containing numbers-at-length.

When defining the tag-recapture time series, the following are also required:

- 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 the expected number of tagged fish in the observation by the detection probability,
- the tagged category or categories (Make up the recaptures),
- the categories scanned (All the fish sampled for tags),
- the length bins if the observations are length-based in an age-based model,
- The selectivities for the categories.

An example of a tag recapture observation:

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

# individuals tagged in 2011 and recaptured in 2012 in Area1
@observation Tag_2011_Area1_recap_2012
type tag_recapture_by_length
# scanned categories in Area1
categories format=*.Area1.*+
```

```
# male and femaled tagged categories
tagged_categories *.Area1.2011+
detection 0.85 ## detection probability
likelihood binomial
selectivities One
tagged_selectivities One
# years to apply observation
years 2012
time_step step2
# proportion of mortality applied before observation is calculated
time_step_proportion 0.5

table scanned
2012 281271 41360 30239 12234
end_table

table recaptured
2012 15 20 12 2
end_table

# robustification value to prevent divide by zero errors
delta 1e-11
# Likelihood dispersion
dispersion 6.3
```

The observed ($O_{y,l}$) and expected ($E_{y,l}$) values in year y and length l of this observation are:

$$O_{y,l} = \frac{R_{y,l}}{S_{y,l}} \tag{7.4}$$

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

$$E_{y,l} = d\frac{\tilde{N}_{y,l,t} + (\tilde{N}_{y,l,t+1} - \tilde{N}_{y,l,t}) \times p}{N_{y,l,t} + (N_{y,l,t+1} - N_{y,l,t}) \times p} \tag{7.5}$$

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 observation occurred, p is the proportion of the time step that the observation was taken, and d is the detection probability.

For observations with multiple tagged categories and multiple categories that were vulnerable to sampling:

$$\tilde{N}_{y,l,t} = \sum_{j=1}^{J} N_{y,l,t,j} \tag{7.6}$$

where $j = \{1,2,3,...,J\}$ are all the tagged categories, the same method is applied to the vulnerable categories to calculate $N_{y,l,t}$. The tagged categories should be defined in the vulnerable categories. In an extreme case where every individual in the population is tagged, this result would be divided by zero. So, to constrain the expectation to be between 0 and 1, the numerator must be in the denominator.

The tag-recapture likelihood (binomial) is specified below. 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 $c.v.$, although there are user-specified robustification and dispersion parameters available. The factorials are calculated using the log-gamma function, to allow for non-integer arguments where necessary (and to avoid overflow errors).

Tag Recapture by length for growth

Designed for situations where you intend to estimate growth using the size frequency of recaptured fish, without using any information on scanned fish. This replicates CASALs tag recapture observation `type=growth`.

An example of a tag recapture observation:

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


@observation 1997_recaptures3_1
type tag_recapture_by_length_for_growth
years 1997
categories male.2011 + female.2011
selectivities One
likelihood multinomial
delta 1e-11
time_step May-Sep
time_step_proportion 0.5
table recaptured
1997 1 2 2 3 2 3 2 3 4
end_table
```

The observed ($O_{y,l}$) and expected ($E_{y,l}$) values in year y and length l of this observation are:

$$O_{y,l} = R_{y,l} \frac{R_{y,l}}{\sum_l R_{y,l}} \tag{7.7}$$

where $R_{y,l}$ is the number of recaptures in year y at length l.

The expected value derived from Casal2 uses interpolation over the mortality block. The categories defined in the configuration files are calculated over this block as follows.

$$N_{y,l,c} = N^{pre}_{y,l,c} + (N^{post}_{y,l,c} - N^{pre}_{y,l,c})p \tag{7.8}$$

where, p is the proportion of the time step that the observation was taken, $N^{pre}_{y,l,c}$ is the partition before the mortality block and $N^{post}_{y,l,c}$ is the partition after the mortality block.

$$E_{y,l,c} = \frac{N_{y,l,c}}{\sum_c \sum_l N_{y,l,c}} \tag{7.9}$$

where $N_{y,l,c}$ is the number of tagged fish in category c for year y and length bin l.

Note that if you are also applying fishing or natural mortality to the population between the time of release and recapture, then the number of fish at size may be biased (although this may depend on the various mortality or fishing selectivities applied). A work-around is to define the tagged fish as being from a separate stock in a separate area (i.e., where they are not subject to fishing mortality, and the natural mortality is either zero or applied as a constant rate over all length classes).

7.1.2 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 expected values as the mortality block versions described in Section 7.1.1. With the exception that instead of wrapping a mortality block they can wrap any process type available in Casal2.

7.1.3 Specific process observations

A list of types that are associated with this set of observations are:

- process_removals_by_length

Process removals by length

Process-Removals-at-length observations are observations of the relative length frequency part way through a process of type mortality_instantaneous. This observation is exclusively associated with the process of type mortality_instantaneous, and will produce an error if it is associated with any other process type.

The observation is supplied for a given year and time-step, for selected length classes of exploited categories.

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

$$E[N_{l,c,m}] = N_{l,c}U_{l,m}S_{l,c,m}0.5M_{l,c} \tag{7.10}$$

where $N_{l,c}$ are the numbers-at-age in category c before the process is executed, $U_{l,m}$ is the exploitation rate for length bin l from method m, $S_{l,c,m}$ is the selectivity, and M is the natural mortality.

The observation class accesses $E[N_{l,c,m}]$ from the process then the observations are aggregated by method and category depending on how the user specifies the observation, before converting numbers-at-length to proportions-at-length and then calculating the likelihood.

Likelihoods that are available for this observation class are the multinomial, Dirichlet, and the lognormal. See Section 7.2 for information on the respected likelihood.

7.2 Likelihoods

7.2.1 Likelihoods for composition observations

Casal2 has a range of likelihoods for composition observations, these include the multinomial, Dirichlet, Dirichlet-Multinomial, and the lognormal likelihood. Composition observations consist of proportions at age or length. The following notation uses b to denote a composition bin which can be interpreted as an age or length bin.

The multinomial likelihood

For the observed proportions at age O_b for composition bin b (age or length), with sample size N, and the expected proportions for the same bin denoted by E_b, the negative log-likelihood is:

$$-\log(L) = -\log(N!) + \sum_b \log((NO_b)!) - NO_b \log(Z(E_b, \delta)) \tag{7.11}$$

where $\sum_b O_b = 1$ and $\sum_b E_b = 1$. $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$.

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.12}$$

The default value of δ is 1×10^{-11}.

The Dirichlet likelihood

For the observed proportions at age O_b for composition bin b (age or length), with sample size N, and the expected proportions for the same bin denoted by E_b, the negative log-likelihood is:

$$-\log(L) = -\log(\Gamma \sum_b (\alpha_b)) + \sum_b \log(\Gamma(\alpha_b)) - \sum_b (\alpha_b - 1) \log(Z(O_b, \delta)) \tag{7.13}$$

where $\alpha_b = Z(NE_b, \delta)$, $\sum_b O_b = 1$, and $\sum_b E_b = 1$. $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$.

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.14}$$

The default value of δ is 1×10^{-11}.

The Dirichlet multinomial likelihood

The Dirichlet multinomial can be applied using the linear re-parametrised approach from (Thorson et al., 2017). For the observed proportions O_b for composition bin b (age or length), with sample size N, expected proportions for the same bin denoted by E_b, and estimable overdispersion parameter θ, the negative log-likelihood is:

$$\begin{aligned} -\log(L) = &-\log\Gamma(N+1) + \sum_b \log(\Gamma(NO_b + 1)) + \\ &\log\Gamma(\theta N) + \log\Gamma(N + \theta N) NO_b - \sum_b \log(NO_b + \theta NE_b) - \log(\theta NE_b) \end{aligned}$$

which has an effective sample size n_{eff}

$$n_{eff} = \frac{1 + \theta N}{1 + \theta} = \frac{1}{1 + \theta} + N \frac{\theta}{1 + \theta}$$

where the effective sample size is a linear function of input sample size with intercept $(1 + \theta)^{-1}$ and slope $\frac{\theta}{1+\theta}$. Casal2 will report the n_{eff} in the observation report under the column label `adjusted_error`.

Interpreting θ:

- if θ is large then $n_{eff} \to N$
- if $\theta \ll N$ and $N > 1$ then θ can be interpreted as the ratio of the effective sample size over the input sample size.

If you estimate θ it is recommended to apply a transformation such as log (see below for example syntax), Casal2 will error out if there is not transformation applied to θ. This likelihood is quite different to configure compared with other likelihood types because it has an estimable parameter you need to define a `@likelihood` block and it cannot have a `label` that is the same as a `type` from any of the other likelihoods.

```
@likelihood DirichletMultinomialFisheryAge
type dirichlet_multinomial
theta 1

@observation FisheryAge
type proportions_at_age
...
likelihood DirichletMultinomialFisheryAge

@parameter_transformation log_theta
type log
parameter likelihood[DirichletMultinomialFisheryAge].theta
```

The lognormal likelihood

For the observed proportions at age O_b for bin b, with c.v. c_b, and the expected proportions at the same age classes E_b, the negative log-likelihood is defined as;

$$-\log(L) = \sum_b \left(\log(\sigma_b) + 0.5 \left(\frac{\log(O_b / Z(E_b, \delta))}{\sigma_b} + 0.5\sigma_b \right)^2 \right) \tag{7.15}$$

where

$$\sigma_b = \sqrt{\log\left(1 + c_b^2\right)} \tag{7.16}$$

and the c_b's are the c.v.s for each composition bin b, and $Z(\theta, \delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$.

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.17}$$

The default value of δ is 1×10^{-11}.

7.2.2 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 qs:

- The qs can be treated as 'nuisance' parameters. For each set of values of the free parameters, the model uses the values of the qs which minimise the objective function. These optimal qs are calculated algebraically (see Section 7.4.2). If one of the qs 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;

- The qs can be treated as ordinary free parameters.

For both options, it is necessary to evaluate the contribution of O_i to the negative log likelihood for a given value of q. Each observation O_i varies about qE_i, which expresses 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 the lognormal is

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i / qZ(E_i, \delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{7.18}$$

where

$$\sigma_i = \sqrt{\log\left(1 + c_i^2\right)} \tag{7.19}$$

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.20}$$

The default value of δ is 1×10^{-11}.

This formulation 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.

The normal likelihood

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

$$-\log(L) = \sum_i \left(\log(c_i E_i) + 0.5 \left(\frac{O_i - E_i}{Z(c_i E_i, \delta)} \right)^2 \right) \tag{7.21}$$

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.22}$$

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.

7.2.3 Likelihoods for tag recapture by age and length observations

The binomial likelihood This likelihood is for situations where the length frequencies or age frequencies of both recaptured tagged fish and of the scanned fish are known. Available in both age or length based models.

The likelihood is defined as a binomial, and based on lengths or ages for both the tag recaptures and scanned individuals.

$$\begin{aligned} -\log(L)' = -\sum_i [\, & \log(n_i!) - \log((n_i - m_i)!) - \log((m_i)!) + m_i \log\left(Z\left(\frac{M_i}{N_i}, \delta\right)\right) \\ & + (n_i - m_i) \log\left(Z\left(1 - \frac{M_i}{N_i}, \delta\right)\right) \,] \end{aligned} \tag{7.23}$$

where

n_i = number of fish at length or age i that were scanned

m_i = number of fish at length or age i that were recaptured

N_i = number of fish at length or age i in the available population (tagged and untagged)

M_i = number of fish at length 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.

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

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

If an over-dispersion parameter (τ) is specified then the final negative log likelihood $-log(L)$ contribution is

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

Note that the over-dispersion is mathematically equivalent to the inverse of a likelihood multiplier on the final negative log-likelihood value, and hence either can be used to achieve the same effect.

7.2.4 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

$$\begin{aligned} -\log(L) = -\sum_i [\, & \log(N_i!) - \log((N_i(1 - O_i))!) - \log((N_i O_i)!) + N_i O_i \log(Z(E_i, \delta)) \\ & + N_i(1 - O_i) \log(Z(1 - E_i, \delta)) \,] \end{aligned} \tag{7.24}$$

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta / (2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{7.25}$$

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;

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{7.26}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{7.27}$$

The default value of δ is 1×10^{-11}.

The Poisson likelihood

For observed value O_i and expected value E_i the negative log-likelihood is defined as;

$$-\log(L) = -\left(-Z(E_i,\delta) + O_i log(Z(E_i,\delta)) - log\Gamma(O_i+1)\right) \tag{7.28}$$

where $Z(\theta,\delta)$ is a robustifying function to prevent log of zero errors, with parameter $\delta > 0$.

7.3 Data weighting

Data weighting can be modified by the use of subcommands for each observation. This can be defined for observations using either process error, an error value multiplier, a likelihood multiplier, or (in the case of tag recaptures) an overdispersion parameter. The different methods have the effect of up-weighting or down-weighting the likelihood contribution for a specific observation.

Process errors (`process_errors`) increase the observation error in the data, and hence of decreasing the relative weight given to the data. The method by which process errors modify the error values depends on the likelihood type (see below).

The error value multiplier modifies the error value (e.g., the N or CV depending on the likelihood) simply by multiplying it by a constant. Similarly, the likelihood multiplier modifies the observation likelihood by multiplying it by a constant.

In the case of tag recaptures, an overdispersion parameter can be applied to modify the error values. This is mathematically equivalent to multiplying the likelihood value by the inverse of the dispersion value.

For observations where the likelihood is parameterised by the CV, the process error can be specified for a given set of observations as a CV, in which case all the CVs c_i are changed to

$$c_i' = \sqrt{c_i^2 + c_{process\_error}^2} \tag{7.29}$$

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,

$$N_i' = \frac{1}{1/N_i + 1/N_{process\_error}} \tag{7.30}$$

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

For both the CV and N process errors, the process error has more effect on small errors than on large ones. Note that a large value for the N process error means a small process error.

Note that `process_errors` is not allowed for tag recapture observations — use the dispersion subcommand instead.

7.4 Catchability q parameters

Catchability parameters often denoted by q are used in abundance or biomass observations to scale the model expected value to the observed value (see Section 7.1.1). Casal2 has two methods for implementing catchability parameters, `free` and `nuisance`.

The `free` approach treats the catchability parameters like all other parameters, where a prior needs to be assumed and it is estimated in the usual fashion.

The `nuisance` approach treats the catchability parameters like all other parameters, where a prior needs to be assumed and it is estimated in the usual fashion.

7.4.1 Free q

(see Section 10.5.1 for additional details on syntax)

```
@catchability CPUEq
type free
q 0.1
```

7.4.2 Nuisance q

This section describes the algorithms Casal2 uses to derive nuisance (analytical) catchability coefficients qs (see Section 10.5.2 for additional details on syntax). From the user's point of view, the essence is that you can use nuisance qs in the following situations:

- With maximum likelihood estimation
- With Bayesian estimation, providing that the additional prior on q is one of the following:
  - None (default)
  - Uniform-log
  - Lognormal with observations distributed lognormal, robustified lognormal

The scenarios in which the nuisance catchability q can be used in a Bayesian analysis (Table 7.1):

Table 7.1: Equations used to calculate nuisance qs. (*=no analytic solution found.)

Distribution	Maximum Likelihood	None	Uniform-log	Normal	lognormal
Normal	(7.31)	(7.31)	(7.33)	*	*
Lognormal	(7.34)	(7.34)	(7.38)	*	(7.39)

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

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

$$-log(L) = \sum_i log(c_i q E_i) + 0.5 \sum_i \left(\frac{O_i - qE_i}{c_i q E_i} \right)^2 \tag{7.31}$$

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

$$\frac{\partial}{\partial q}(-log(L)) = \frac{n}{q} + \frac{1}{q^2} \sum_i \frac{O_i}{c_i^2 E_i} - \frac{1}{q^3} \sum_i \left(\frac{O_i}{c_i E_i} \right)^2 \tag{7.32}$$

hence

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4nS_2}}{2n} \tag{7.33}$$

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,

$$-log(L) = \sum_i log(\sigma_i) + 0.5 \sum_i \left(\frac{log(O_i) - log(qE_i) + 0.5\sigma_i^2}{\sigma_i} \right)^2 \tag{7.34}$$

$$\frac{\partial}{\partial q}(-log(L)) = \frac{-1}{q} \sum_i \left(\frac{log(O_i/E_i) - log(q) + 0.5\sigma_i^2}{\sigma_i^2} \right) \tag{7.35}$$

$$\hat{q} = exp \frac{0.5n + S_3}{S_4} \tag{7.36}$$

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 a prior for q must be specified.

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, respectively, and $\sigma_q = \sqrt{log(1 + c_q^2)}$. Since the prior is uniform, the equation for $\hat{q}$ is the same as the maximum likelihood estimation.

When the (O_i) are assumed to be normally distributed and the prior is log-uniform equation (7.33) becomes,

$$\hat{q} = \frac{-S_1 + \sqrt{S_1^2 + 4(n+1)S_2}}{2(n+1)} \tag{7.37}$$

but $\hat{q}$ with either a Normal or Lognormal prior cannot be solved for.

When the O_i are assumed to be Lognormally distributed and the prior is log-uniform, equation (7.36) becomes

$$\hat{q} = exp \frac{0.5n - 1 + S_3}{S_4} \tag{7.38}$$

and if the prior is Lognormal,

$$\hat{q} = exp\frac{0.5n - 1.5 + log(\mu_q)/\sigma_q^2 + S_3}{S_4 + 1/\sigma_q^2} \tag{7.39}$$

However, it is not possible to solve for $\hat{q}$ with a normal prior.

An example of specifying the syntax and an equivalent additional prior

```
@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
```

7.5 Simulating observations

Casal2 can generate simulated observations for a given model with a set of parameter values using `casal2 -s n` to simulate n sets of observations). Simulated observations are randomly generated values, which are generated with the error distributions 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 use the 'true' parameter values which are fed via the `-i/-I` file input which generate expected values. Then, if a set of observations use 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 the simulated values, have three components which the user can change. These are the (i) the likelihood choice and observation error, (ii) parameter uncertainty through the use of `-i/-I`, and (iii) time-varying parameters.

- 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 $E_i c_i$.
- 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.
- 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,
  1. 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.
  2. Each simulated observation value S_i is calculated as the proportion of the N sampled values equalling i
  3. The simulated observation values S_i are then rescaled so that their sum is equal to 1
- 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,

1. A sample of N_i independent binary variates is generated, equalling 1 with probability E_i
2. 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 using a `casal2 -s 1 -i input_pars.out` run. A report must be defined using the `simulated_observation` report (`@report[label].type=observation`). For completeness the report is described along with some best practices here, but there is additional information in Section 8.

A typical report for simulating an observation looks like

```
@report CPUE_index_sim # report label
type simulated_observation # report tyoe
observation CPUEandes # observation to simulate
file_name sim/CPUEandes # file to write simualted data to
```

note that in the subcommand `file_name` there is a directory component `sim`. It is recommended when doing simulations that you create directories that can be documented on what configurations caused that set of simulated datasets. This will become useful if you are looking at multiple simulated models assumptions.

Simulated reports will be produced with the following extension `.1_1`. The first number of the extension relates to the row of the `-i`/`-I` file and the second number (separated by `_`) represents the simulation iteration defined by the `n` argument in the configuration input `casal2 -s n`. Examples of the extension follow,

- `.1_1` indicates simulated data produced from the first row of parameters and is the first random draw
- `.1_2` indicates simulated data produced from the first row of parameters and is the second random draw
- `.2_10` indicates simulated data produced from the second row of parameters and is the 10^{th} random draw

7.6 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 be generated only if a report command `@report[label].type=observation` is specified.
- The observed values should be supplied (even if they are 'dummy' observations). These observation values will be processed by Casal2 as if they were actual observation values, and must be in the same format as actual observation values.
- The subcommands `likelihood`, `obs`, `error_value`, and `process_error` have no effect when generating the expected values for the pseudo-observation.
- When simulating observations, the subcommand `simulation_likelihood` to indicate the likelihood to use. In this case, the `obs`, `error_value`, and `process_error` are used to determine the appropriate terms to use for the likelihood when simulating.

7.7 Residuals

Casal2 will print the default residual values (i.e., observed less fitted) only when the report type `@report.type=observation` is used. 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 the Casal2 **R** package. For specific **R** functions see Section 17.

The definitions used in the calculations are

- *Pearson residuals* attempt to express the residual relative to the variability of the observation, and are defined as (O-F)/std.dev.(O), where std.dev.(O) is calculated as
  - F $\times$ cv for normal, lognormal, robustified lognormal, and normal-log error distributions.
  - s for normal-by-standard deviation error distributions.
  - $\sqrt{\frac{Z(F,r)(1-Z(F,r))}{N}}$ for multinomial or binomial likelihoods.
  - $\sqrt{\frac{(F+r)(1-F+r)}{N}}$ for binomial-approx likelihood likelihoods.
- *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 function is described earlier in the likelihood section.

8 The report section: output and reports

Casal2generates a range of outputs, both information and error messages, and reports that summarise the model outputs. The report section is where the reports are defined that are used to summarise and print model output.

The command and subcommand syntax for the report section is given in Section 12.1.

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

8.1 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. The most useful report is the default report (see Section 8.3) which will auto generate many of the reports users require.

```
@report default
type default
selectivities true
derived_quantities true
observations true
processes true
catchabilities true
time_varying true
parameter_transformations true
time_varying true


@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., ...
```

8.2 Report output format

Reports from Casal2 have a standard style (with the exception of `output_parameters` and `simulated_observation`, see below). The standard style is that reports are prefixed with an asterisk 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)
...
... # report content
...
*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 time step 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.

8.3 Print default reports

This is a report type that generates a range of other default reports, The report queries model and generates default reports for all catchabilities, observations, processes, selectivities, derived parameters, time-varying parameters, parameter transformations, and projections. This report will print out in run modes `-r, -e, -f`.

```
@report default
type default
selectivities true
derived_quantities true
observations true
processes true
catchabilities true
time_varying true
parameter_transformations true
time_varying true
```

8.4 Print a summary of an initialisation step

This report prints a summary of an initialisation, including convergence statistics for iterative initialisations. This report will print out in run modes `-r, -e, -f`.

8.5 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`.

8.6 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 time step. This report is evaluated at the end of the time step in the given year(s). This report will print out in run modes `-r, -e, -f`.

8.7 Print the partition biomass

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

8.8 Print the values(s) of an addressable

This report prints the value or values of an addressable for each given year or given years and time step. This report is evaluated at the end of the time step in the given year(s). This report will print out in run modes `-r, -e, -f`.

8.9 Print the Growth models

This report prints the length and weight relationship as well as growth increment model used to move fish along the length partition. This will print the growth transition matrix and other important values. This report is evaluated at the end of the time step in the given year(s). This report will print out in run modes `-r, -e, -f`.

```
@report growth_increment_model
type growth_increment
time_step step2
years 1900:2013
growth_increment growth_model
```

8.10 Print a parameter transformation

This report prints a specific `@parameter_transformation` block with the values This report will print out in run modes `-r, -e, -m`. If you have many transformations it is best to report these using the default report see Section 8.3

```
@report log_b0
type parameter_transformation
parameter_transformation log_b0
```

8.11 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`.

8.12 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`.

8.13 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`.

8.14 Print the estimate values (the free parameters in the free parameter file 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`.

8.15 Print the output parameters

This report prints the output (i.e., the free) parameters and their values as a simple table. This report will print out in run modes `-r, -e, -f`.

8.16 Print the result of an estimation

This report prints a summary of the minimisers convergence or success status from an estimation. This report will print out in run modes `-e, -p`.

8.17 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`.

8.18 Print the covariance matrix

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

8.19 Print the correlation matrix

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

8.20 Print the Hessian matrix

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

8.21 Print the catchability values

This report prints the catchability for a requested catchability.

8.22 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
```

8.23 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`.

Simulated reports will be produced with the following extension `.1_1`. The first number of the extension relates to the row of the `-i`/`-I` file and the second number (separated by `_`) represents the simulation iteration defined by the `n` argument in the configuration input `casal2 -s n`. Examples of the extension follow,

- `.1_1` indicates simulated data produced from the first row of parameters and is the first random draw
- `.1_2` indicates simulated data produced from the first row of parameters and is the second random draw
- `.2_10` indicates simulated data produced from the second row of parameters and is the 10^{th} random draw

8.24 Print selectivities

This report prints the values of a selectivity for each length in the partition. See Section 12.1.28 for syntax information.

8.25 Print selectivities by year

This report prints the values of a selectivity for each age/length for the given model years. Useful when you have a model with time-varying selectivity. See Section 12.1.29 for syntax information.

8.26 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.

8.27 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`.

8.28 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`.

8.29 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`.

8.30 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
```

8.31 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 **R** package. An example usage is reading in files of MCMC posterior values of derived quantities, which can then be plotted.

9 Population command and subcommand syntax

The description of the methods for the population section is given in Section 5.

In the following section, the sub-section headers use a notation of the form "**@observation[label].type=abundance**" which, in this case, represents the input command fragment

```
@observation label # where label is a unique label for that observation
type=abundance
...
```

The specific subcommands for a command are given within each command.

9.1 Model structure

`@model` *`label`* Define an object of type *Model*. See Section 5.2 for more information.

`type` Type of model (either type=age or type=length)
Type: String
Default: age

`base_weight_units` Define the units for the base weight measurement unit (grams, kilograms (kgs), or tonnes). This will be the default unit of any weight input values
Type: String
Default: tonnes

`threads` The number of threads to use for this model
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

9.1.1 Model of type Length

`@model[label].type=length`.

`start_year` Define the first year of the model, immediately following initialisation
Type: Non-negative integer
Default: No default
Value: Defines the first year of the model, must be $\geq$ 1000, 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

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

`time_steps` Define the labels of the time steps, in the order that they are applied, to form the

annual cycle
Type: Vector of strings
Default: No default
Value: A list of valid labels defined by `@time_step`

`projection_final_year` Define the final year of the model when running projections
Type: Non-negative integer
Default: 0
Value: A value greater than final_year

`length_bins` The minimum length in each length bin
Type: Vector of real numbers (estimable)
Default: true
Value: $0 \leq \text{length}_{\text{min}} \leq \text{length}_{\text{max}}$

`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: Real number (estimable)
Default: 0
Value: $\text{length}_{\text{max}} <$ length_plus_group

9.2 Initialisation

`@initialisation_phase` *`label`* Define an object of type *Initialisation_Phase*. See Section 5.2.2 for more information.

`label` The label of the initialisation phase
Type: String
Default: No default

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

9.2.1 Initialisation Phase of type Iterative

`@initialisation_phase[label].type=Iterative`. See Section 5.2.2 for more information.

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

`insert_processes` The processes in the annual cycle to be include in this initialisation phase

Type: Vector of strings
Note: To insert a process during initialisation, it needs to subscribe to the following format `time_step_label()=process_label`

`exclude_processes` The processes in the annual cycle to be excluded from this initialisation phase
Type: Vector of strings that are process labels to exclude

`convergence_years` The iteration (year) when the test for convergence (λ) is evaluated
Type: Vector of non-negative integers
Default: true

`lambda` The maximum value of the proportional summed difference between the partition at year and year+1 that indicates successful convergence
Type: Real number
Default: 1e-10

`plus_group` Indicates if the convergence check applies only to the plus_group of the partition
Type: boolean
Default: false

9.3 Categories

`@categories` *`label`* Define an object of type *Categories*. See Section 4.2 for more information.

`format` The format that the category names use
Type: String
Default: No default

`names` The names of the categories
Type: Vector of strings
Default: No default
Note: Category names must start with a letter or underscore

`growth_increment` The growth increment labels for each category
Type: Vector of strings
Default: true
Value: Valid labels of growth-increment relationships

9.4 Time-steps

`@time_step` *`label`* Define an object of type *Time_Step*. See Section 5.2.1 for more information.

`label` The label of the time step
Type: String
Default: No default

`processes` The labels of the processes that occur in this time step, in the order that they occur
Type: Vector of strings
Default: No default

9.5 Processes

`@process` *`label`* Define an object of type *Process*. See Section 5.3 for more information.

`label` The label of the process
Type: String
Default: No default

`type` The type of process
Type: String
Default: No default

9.5.1 Process of type Mortality Constant Rate

`@process[label].type=Mortality_Constant_Rate`. See Section 5.3.2 for more information.

`categories` The list of category labels
Type: Vector of strings
Default: No default

`m` The mortality rates
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`time_step_proportions` The time step proportions for the mortality rates
Type: Vector of real numbers
Default: false
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)
Value: The time step proportions must sum to one. If only one value is supplied, then the each time step is allocated an equal proportion. Otherwise the number of values must equal the number of time steps

`relative_m_by_length` The list of mortality by length ogive labels for the categories
Type: Vector of strings
Default: No default

9.5.2 Process of type Mortality Constant Exploitation

`@process[label].type=Mortality_Constant_Exploitation`. See Section 5.3.2 for more information.

`categories` The list of category labels

Type: Vector of strings
Default: No default

`u` The exploitation rates
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`time_step_proportions` The time step proportions for the exploitation rates
Type: Vector of real numbers
Default: false
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)
Value: The time step proportions must sum to one. If only one value is supplied, then the each time step is allocated an equal proportion. Otherwise the number of values must equal the number of time steps

`relative_u_by_length` The list of exploitation by length ogive labels for the categories
Type: Vector of strings
Default: No default

9.5.3 Process of type Mortality Disease Rate

`@process[label].type=Mortality_Disease_Rate`. See Section 5.3.2 for more information.

`categories` The list of category labels
Type: Vector of strings
Default: No default

`disease_mortality_rate` The disease mortality rates
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 10.0 (inclusive)

`year_effects` Annual deviations around the disease mortality rate
Type: Vector of real numbers (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`selectivities` The list of selectivities
Type: Vector of strings
Default: No default

`years` Years in which to apply the disease mortality in
Type: Vector of non-negative integers
Default: No default

9.5.4 Process of type Mortality Instantaneous

`@process[label].type=Mortality_Instantaneous`. See Section 5.3.2 for more information.

`categories` The categories for instantaneous mortality
Type: Vector of strings
Default: No default

`m` The natural mortality rates for each category
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`time_step_proportions` The time step proportions for natural mortality
Type: Vector of real numbers
Default: true
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)
Value: Proportions must sum to one

`biomass` Switch to indicate if the catches are biomasses or abundances
Type: Boolean
Default: True

`relative_m_by_length` The M-by-length selectivities to apply to each of the categories for natural mortality
Type: Vector of strings
Default: No default

`table` The table of data specifying the catches for each fishery, the categories, years, and the U_max
Type: Data table with label =
Default: No default
Value:
Note: See 16.2 for more details on specifying data tables

9.5.5 Process of type null process

`@process[label].type=null_process`.

The null_process type has no additional subcommands. Note that this process does nothing. It is included primarily as a means of replacing other processes with ”no action” to allow for testing of alternative model structures.

9.5.6 Process of type Recruitment Beverton Holt

`@process[label].type=Recruitment_Beverton_Holt`. See Section 5.3.1 for more information.

`categories` The category labels
Type: Vector of strings
Default: No default

`r0` R0, the mean recruitment used to scale annual recruits or initialise the model
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Value: Use either R0 or B0, but not both

`b0` B0, the SSB corresponding to R0, and used to scale annual recruits or initialise the model
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Value: Use either R0 or B0, but not both

`proportions` The proportion for each category
Type: Real number (estimable)
Default: No default

`initial_mean_length` The initial mean length at recruitment
Type: Non-negative real number (estimable)
Default: No default

`initial_length_cv` The initial length cv at recruitment
Type: Non-negative real number (estimable)
Default: No default

`ssb_offset` The spawning biomass year offset
Type: Non-negative integer
Default: No default

`steepness` Steepness (h)
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.2 (inclusive)
Upper bound: 1.0 (inclusive)

`ssb` The SSB label (i.e., the derived quantity label)
Type: String
Default: No default

`b0_initialisation_phase` The initialisation phase label that B0 is from
Type: String
Default: No default

`ycs_values` Deprecated
`ycs_years` Deprecated
`standardise_ycs_years` Deprecated

`recruitment_multipliers` The recruitment values also termed year class strengths.
Type: Vector of real numbers (estimable)
Default: No default

`standardise_years` The years that are included for year class standardisation, they refer to the recruited year not spawning or year class year.
Type: Vector of non-negative integers
Default: true

9.5.7 Process of type Recruitment Constant

`@process[label].type=Recruitment_Constant`. See Section 5.3.1 for more information.

`categories` The categories
Type: Vector of strings
Default: No default

`proportions` The proportion for each category
Type: Real number (estimable)
Default: true

`length_bins` The length bin that recruits are uniformly distributed over at the time of recruitment
Type: Non-negative integer
Default: No default

`r0` R0, the recruitment used for annual recruits and initialise the model
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

9.5.8 Process of type Tagging

`@process[label].type=Tagging`. See Section 5.3.4 for more information.

`from` The categories that are selected for tagging (i.e, transition from)
Type: Vector of strings
Default: No default

`to` The categories that have tags (i.e., transition to)
Type: Vector of strings
Default: No default

`penalty` The penalty label

Type: String
Default: No default

`u_max` The maximum exploitation rate (U_max)
Type: Real number
Default: 0.99
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (exclusive)

`compatibility_option` Backwards compatibility option: either casal2 (the default) or casal. This affects the penalty and age-length calculations
Type: string
Default: casal2
Value: Valid options are `casal2` & `casal`

`years` The years to execute the tagging events in
Type: Vector of non-negative integers
Default: No default

`initial_mortality` The initial mortality to apply to tags as a proportion
Type: Real number (estimable)
Default: 0.0
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`initial_mortality_selectivity` The initial mortality selectivity label
Type: String
Default: No default
Value: A valid selectivity label

`selectivities` The selectivity labels
Type: Vector of strings
Default: No default
Value: Valid selectivity labels

`n` The total number of tags to apply
Type: Vector of real numbers (estimable)
Default: No default
Note: Only required if table proportions are also supplied

`table` The table of data specifying the either the numbers or proportions to tag from and to each category and year
Type: Data table with either label = numbers or label = proportions
Default: No default
Note: If table proportions, then the total number (@process[label].n) should also be specified. See 16.2 for more details on specifying data tables

`table numbers` The table of releases as numbers for the process
Type: Data table with label = numbers
Default: Can be replaced with 'table proportions' – see below
Value: A $n_y \times (n_l \times n_c) + 1$ matrix, where $n_y =$ is the number of years, n_l are the number of length bins and n_c are the number of categories. The first column is the year value for that row. See below for an example
Note: example below

```
table numbers
1993 34 34 23 43
1994 23 34 23 43
end_table
```

`table proportions` The table of releases as numbers for the process
Type: Data table with label = proportions
Default: Can be replaced with table numbers – see above
Value: A $n_y \times (n_l \times n_c) + 1$ matrix, where $n_y =$ is the number of years, n_l are the number of length bins and n_c are the number of categories. The first column is the year value for that row. See below for an example
Note: example below

```
n 200 300 ## need to specify n if you give proportions
table proportions
1993 0.1 0.2 0.7
1994 0.1 0.2 0.7
end_table
```

`tolerance` Tolerance for checking the specified proportions sum to one
Type: Real number
Default: 1e-5
Lower bound: 0 (inclusive)
Upper bound: 1.0 (inclusive)

9.5.9 Process of type Transition Category

`@process[label].type=Transition_Category`. See Section 5.3.3 for more information.

`from` The categories to transition from
Type: Vector of strings
Default: No default
Value: Valid category labels

`to` The categories to transition to
Type: Vector of strings
Default: No default
Value: Valid category labels

`proportions` The proportions to transition for each category

Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`selectivities` The selectivities to apply to each proportion
Type: Vector of strings
Default: No default
Value: Valid selectivity labels

`include_in_mortality_block` Include this process within the mortality block
Type: Boolean
Default: false
Value: Either true or false
Note: Warning, if true, this may adversely effect derived quantities and observations that interpolate over the mortality block - see See Section 5.3.3 for more information.

9.6 Time varying parameters

`@time_varying` *`label`* Define an object of type *Time_Varying*. See Section 5.10 for more information.

`label` The label of the time-varying object
Type: String
Default: No default

`type` The type of the time-varying object
Type: String
Default: No default

`parameter` The name of the parameter to vary in each year
Type: String
Default: No default

`years` The years in which to vary the parameter
Type: Vector of non-negative integers
Default: No default

9.6.1 Time Varying of type Annual Shift

`@time_varying[label].type=Annual_Shift`. See Section 5.10.4 for more information.

`a` Parameter A
Type: Real number (estimable)
Default: No default

`b` Parameter B

Type: Real number (estimable)
Default: No default

`c` Parameter C
Type: Real number (estimable)
Default: No default

`scaling_years` The scaling years
Type: Vector of non-negative integers
Default: The years in which to vary the parameter

`values` The values
Type: Vector of real numbers (estimable)
Default: No default

9.6.2 Time Varying of type Constant

`@time_varying[label].type=Constant`. See Section 5.10.1 for more information.

`values` The value to assign to addressable
Type: Vector of real numbers (estimable)
Default: No default

9.6.3 Time Varying of type Exogenous

`@time_varying[label].type=Exogenous`. See Section 5.10.3 for more information.

`a` The shift parameter
Type: Real number (estimable)
Default: No default

`exogenous_variable` The values of exogenous variable for each year
Type: Vector of real numbers (estimable)
Default: No default

9.6.4 Time Varying of type Linear

`@time_varying[label].type=Linear`. See Section 5.10.2 for more information.

`slope` The slope of the linear trend (i.e., the additive amount per year)
Type: Real number (estimable)
Default: No default

`intercept` The intercept of the linear trend (, i.e. the value in the first year)
Type: Real number (estimable)
Default: No default

9.6.5 Time Varying of type Random Draw

`@time_varying[label].type=Random_Draw`. See Section 5.10.6 for more information.

`mean` The mean (μ) of the random draw distribution
Type: Real number (estimable)
Default: 0

`sigma` The standard deviation (σ) of the random draw distribution
Type: Real number (estimable)
Default: 1.0
Value: A positive real number

`lower_bound` The lower bound for the random draw
Type: Real number (estimable)
Default: No default

`upper_bound` The upper bound for the random draw
Type: Real number (estimable)
Default: No default

`distribution` The distribution type
Type: String
Default: normal
Value: Only the normal and lognormal are implemented

9.6.6 Time Varying of type Random Walk

`@time_varying[label].type=Random_Walk`. See Section 5.10.5 for more information.

`mean` The mean (μ) of the random walk distribution
Type: Real number (estimable)
Default: 0.0

`sigma` The standard deviation (σ) of the random walk distribution
Type: Real number (estimable)
Default: 1.0
Value: A positive real number

`lower_bound` The lower bound for the random walk
Type: Real number (estimable)
Default: No default

`upper_bound` The upper bound for the random walk
Type: Real number (estimable)
Default: No default

`rho` The autocorrelation parameter (ρ) of the random walk distribution

Type: Real number (estimable)
Default: 1

`distribution` The distribution type
Type: String
Default: normal
Value: Only the normal distribution is implemented

9.7 Derived quantities

`@derived_quantity` *`label`* Define an object of type *Derived_Quantity*. See Section 5.4 for more information.

`label` The label of the derived quantity
Type: String
Default: No default

`type` The type of derived quantity
Type: String
Default: No default

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

`categories` The list of categories to use when calculating the derived quantity
Type: Vector of strings
Default: No default

`selectivities` The list of selectivities to use when calculating the derived quantity
Type: Vector of strings
Default: No default

`time_step_proportion` The proportion through the mortality block of the time step when the derived quantity is calculated
Type: Real number (estimable)
Default: 0.5
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`time_step_proportion_method` The method for interpolating for the proportion through the mortality block
Type: String
Default: weighted_sum
Value: weighted_sum or weighted_product. weighted_sum is usually the most sensible if using instantaneous mortality

9.7.1 Derived Quantity of type Abundance

`@derived_quantity[label].type=Abundance`. See Section 5.4 for more information.

The Abundance type has no additional subcommands.

9.7.2 Derived Quantity of type Biomass

`@derived_quantity[label].type=Biomass`. See Section 5.4 for more information.

`age_weight_labels` The labels for the age-weights that correspond to each category for the biomass calculation
Type: Vector of strings
Default: No default
Value: If age-weights are not used, do not use the subcommand

9.7.3 Derived Quantity of type Biomass Index

`@Derived_Quantity[label].type=biomass_index`. See Section 5.4 for more information.

`age_weight_labels` The labels for the age-weights that correspond to each category for the biomass calculation
Type: Vector of strings
Default: No default
Value: If age-weights are not used, do not use the subcommand

`distribution` The type of distribution for the biomass index
Type: String
Default: lognormal
Value: Either `normal`, `lognormal`, or `none`

`cv` The cv for the uncertainty for the distribution when generating the biomass index
Type: Real number
Default: 0.2
Lower bound: 0.0 (exclusive) for the `normal` or `lognormal`

`bias` The bias (a positive or negative proportion) when generating the biomass index
Type: Real number
Default: 0.0

`rho` The autocorrelation in annual values when generating the biomass index
Type: Real number
Default: 0.0
Lower bound: 0.0 (inclusive)

`catchability` The catchability coefficient to use when generating the biomass index
Type: String
Default: None
Value: Either `none` or a valid catchability label. If none, then no catchability multiplier is applied

9.8 Growth-Increment

`@growth_increment` *`label`* Define an object of type *growth_increment*. See Section 5.6 for more information.

`label` The label of the growth increment model
Type: String
Default: No default

`type` The type of growth increment model
Type: String
Default: No default

`time_step_proportions` The proportion of annual increment to apply in each time-step. Must sum = 1.0
Type: Vector of real numbers
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`compatibility_option` Backwards compatibility option: either casal2 (the default) or casal to use the (less accurate) cumulative normal function from CASAL
Type: String
Default: casal2
Note: options allowed,`casal`, `casal2`

`distribution` The assumed distribution for the growth curve
Type: String
Default: normal
Note: options allowed,`normal`

`length_weight` The label from an associated length-weight block
Type: String
Default: No default

`cv` The *cv* for the growth model.
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`min_sigma` The minimum standard deviation for the growth model.
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

9.8.1 Growth Increment of type Basic

`@growth_increment[label].type=basic`. See Section 5.6.3 for more information.

`g_alpha` The g_α parameter
Type: Real number (estimable)
Default: No default

`g_beta` The g_β parameter
Type: Real number (estimable)
Default: No default

`l_alpha` The l_α parameter
Type: Real number (estimable)
Default: No default

`l_beta` The l_β parameter
Type: Real number (estimable)
Default: No default

9.8.2 Growth Increment of type Exponential

`@growth_increment[label].type=exponential`. See Section 5.6.2 for more information.

`g_alpha` The g_α parameter
Type: Real number (estimable)
Default: No default

`g_beta` The g_β parameter
Type: Real number (estimable)
Default: No default

`l_alpha` The l_α parameter
Type: Real number (estimable)
Default: No default

`l_beta` The l_β parameter
Type: Real number (estimable)
Default: No default

9.8.3 Growth Increment of type None

`@growth_increment[label].type=none`. Note: Used for testing and validating models

9.9 Length-weight

`@length_weight` *`label`* Define an object of type *Length_Weight*. See Section 5.7 for more information.

`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

9.9.1 Length Weight of type Basic

`@length_weight[label].type=Basic`. See Section 5.7.2 for more information.

`a` The a parameter ($W = aL^b$)
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`b` The b parameter ($W = aL^b$)
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`units` The units for weights (grams, kilograms (kgs), or tonnes)
Type: String
Default: No default

9.9.2 Length Weight of type None

`@length_weight[label].type=None`. See Section 5.7.1 for more information.

The None type has no additional subcommands.

9.10 Selectivities

`@selectivity` *`label`* Define an object of type *Selectivity*. See Section 5.9 for more information.

`label` The label for the 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` The number of quantiles to evaluate a length-based selectivity over the age-length

distribution
Type: Non-negative integer
Default: 5

`values`
Type: Vector of addressables
Default: No default

`length_values`
Type: Vector of addressables
Default: No default

9.10.1 Selectivity of type All Values

`@selectivity[label].type=All_Values`. See Section 5.9.3 for more information.

`v` The v parameter
Type: Vector of real numbers (estimable)
Default: No default

9.10.2 Selectivity of type All Values Bounded

`@selectivity[label].type=All_Values_Bounded`. See Section 5.9.4 for more information.

`l` The low value (L)
Type: Non-negative integer
Default: No default

`h` The high value (H)
Type: Non-negative integer
Default: No default

`v` The v parameter
Type: Vector of real numbers (estimable)
Default: No default

9.10.3 Selectivity of type Constant

`@selectivity[label].type=Constant`. See Section 5.9.1 for more information.

`a` The a value in $ax^b + c$
Type: Real number (estimable)
Default: 0.0
Lower bound: 0.0 (inclusive)
Note: The defaults result in a simple linear constant where $x = 1$ for all values of x

`b` The b value in $ax^b + c$

Type: Real number (estimable)
Default: 0.0
Note: The defaults result in a simple linear constant where $x = 1$ for all values of x

`c` The c value in $ax^b + c$
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (inclusive)
Note: The defaults result in a simple linear constant where $x = 1$ for all values of x

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0.0
Lower bound: 0.0 (inclusive)

9.10.4 Selectivity of type Double Exponential

`@selectivity[label].type=Double_Exponential`. See Section 5.9.12 for more information.

`x0` The $x0$ parameter
Type: Real number (estimable)
Default: No default

`x1` The $x1$ parameter
Type: Real number (estimable)
Default: No default

`x2` The $x2$ parameter
Type: Real number (estimable)
Default: No default

`y0` The $y0$ parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`y1` The $y1$ parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`y2` The $y2$ parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

`alpha` The maximum value of the selectivity

Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0
Lower bound: 0.0 (inclusive)

9.10.5 Selectivity of type Double Normal

`@selectivity[label].type=Double_Normal`. See Section 5.9.9 for more information.

`mu` The mean (μ
Type: Real number (estimable)
Default: No default

`sigma_l` The left-hand variance (sigma_l) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`sigma_r` The right-hand variance (sigma_r) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0
Lower bound: 0.0 (inclusive)

9.10.6 Selectivity of type Double Normal Plateau

`@selectivity[label].type=Double_Normal_Plateau`. See Section 5.9.10 for more information.

`a1` The a1 ($a1$
Type: Real number (estimable)
Default: No default

`a2` The a2 ($a2$

Type: Real number (estimable)
Default: No default

`sigma_l` The left-hand variance (sigma_l) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`sigma_r` The right-hand variance (sigma_r) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0
Lower bound: 0.0 (inclusive)

9.10.7 Selectivity of type Double Normal Stock Synthesis

`@selectivity[label].type=Double_Normal_Stock_Synthesis`. See Section 5.9.11 for more information.

`peak` Age or length of plateau (max selectivity)
Type: Real number (estimable)
Lower bound: 0.0 (exclusive)

`y0` Transformed selectivity for the first age or length bin
Type: Real number (estimable)
Lower bound: -20
Upper bound: 0

`y1` Transformed selectivity for the last age or length bins
Type: Real number (estimable)
Lower bound: -20
Upper bound: 10

`descending` The shape of descending limb in either ages or lengths
Type: Real number (estimable)
Default: No default

`ascending` The shape of ascending limb in either ages or lengths

Type: Real number (estimable)
Default: No default

`width` width of plateau how many ages or lengths are in the plateau
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`l` min age or first length bin
Type: Real number
Default: No default
Lower bound: 0.0 (exclusive)

`l` max age or last length bin
Type: Real number
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.8 Selectivity of type Increasing

`@selectivity[label].type=Increasing`. See Section 5.9.5 for more information.

`l` The low value (L)
Type: Non-negative integer
Default: No default

`h` The high value (H)
Type: Non-negative integer
Default: No default

`v` The v parameter
Type: Vector of real numbers (estimable)
Default: No default

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.9 Selectivity of type Inverse Logistic

`@selectivity[label].type=Inverse_Logistic`. See Section 5.9.7 for more information.

`a50` The age or length where the selectivity is 50%
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0
Lower bound: 0.0 (inclusive)

9.10.10 Selectivity of type Knife Edge

`@selectivity[label].type=Knife_Edge`. See Section 5.9.2 for more information.

`e` The edge value
Type: Real number (estimable)
Default: No default

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0

9.10.11 Selectivity of type Logistic

`@selectivity[label].type=Logistic`. See Section 5.9.6 for more information.

`a50` The age or length where the selectivity is 50%
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`beta` The minimum age/length for which the selectivity applies
Type: Real number (constant)
Default: 0
Lower bound: 0.0 (inclusive)

9.10.12 Selectivity of type Logistic Producing

`@selectivity[label].type=Logistic_Producing`. See Section 5.9.8 for more information.

`l` The low value (L)
Type: Non-negative integer
Default: No default

`h` The high value (H)
Type: Non-negative integer
Default: No default

`a50` The a50 parameter
Type: Real number (estimable)
Default: No default

`ato95` the ato95 parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`alpha` The maximum value of the selectivity
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.13 Selectivity of type Compound Left

`@selectivity[label].type=compound_left`. See Section 5.9.13 for more information.

`a50` The a50 ($a50$
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a_min` The (a_min) parameter

Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`left_mean` The left_mean parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`sigma` The sigma parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.14 Selectivity of type Compound Right

`@selectivity[label].type=compound_right`. See Section 5.9.14 for more information.

`a50` The a50 ($a50$
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a_min` The (a_min) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`left_mean` The left_mean parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`to_right_mean` The to_right_mean parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`sigma` The sigma parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.15 Selectivity of type Compound Middle

`@selectivity[label].type=compound_middle`.

`a50` The a50 ($a50$
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a_min` The (a_min) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`left_mean` The left_mean parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`to_right_mean` The to_right_mean parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`sigma` The sigma parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.16 Selectivity of type Compound All

`@selectivity[label].type=compound_all`. See Section 5.9.16 for more information.

`a50` The a50 ($a50$
Type: Real number (estimable)
Default: No default

`ato95` The age or length between 50% and 95% selective
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a_min` The (a_min) parameter

Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`sigma` The sigma parameter
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

9.10.17 Selectivity of type Multi-Selectivity

`@Selectivity[label].type=multi_selectivity`. See Section 5.9.17 for more information.

`years` The years for which we want to apply the corresponding selectivity in
Type: Vector of integer for all model years to apply corresponding selectivity.
Default: No default

`selectivity_labels` The labels of the selectivities, one for each year
Type: Vector of strings defining the labels of the selectivities to be used for each year
Default: No default

`default_selectivity` The selectivity used in missing years
Type: string
Default: No default

`projection_selectivity` The selectivity used in missing years in projections
Type: string
Default: Defaults to `default_selectivity` if not supplied

9.11 Projections

`@project` *`label`* Define an object of type *Project*. See Section 5.12 for more information.

`label` Label
Type: String
Default: No default

`type` Type
Type: String
Default: No default

`years` Years to recalculate the values
Type: Vector of non-negative integers
Default: No default

`parameter` Parameter to project
Type: String
Default: No default

9.11.1 Project of type Constant

`@project[label].type=Constant`. See Section 5.12.2 for more information.

`values` The values to assign to the addressable
Type: Vector of real numbers
Default: No default

`multiplier` Multiplier that is applied to the projected value
Type: Real number
Default: 1.0
Lower bound: 0.0 (exclusive)
Value: A vector of length 1 (for a constant value for all years), or a vector of length years (for a specific value for each year)

9.11.2 Project of type Multiple Values

`@project[label].type=multiple_values`. See Section 5.12.3 for more information.

`multiplier` Multiplier that is applied to the projected value
Type: Real number
Default: 1.0
Lower bound: 0.0 (exclusive)
Value: A vector of length 1 (for a constant value for all years), or a vector of length years (for a specific value for each year)

`table values` The table of data specifying the projected values to use for each row of the supplied free parameter file (`-i` or `-I`) with one column of data for each projected year
Type: table with label = values
Default: No default
Value: A $n_i \times n_y$ matrix. Where n_i is the number of rows (parameter sets) in the free parameter file. And n_y is the number of projection years defined by the input `years`
Note: example below

```
@project future_disease_rates
type multiple_values
parameter process[dtransition].proportions{disease}
years 2024:2026
table values
# 2024 2025 2026
   0.1  0.2  0.3
   0.3  0.4  0.5
end_table
```

9.11.3 Project of type Empirical Sampling

`@project[label].type=Empirical_Sampling`. See Section 5.12.4 for more information.

`start_year` The start year of sampling

Type: Non-negative integer
Default: No default

`final_year` The final year of sampling
Type: Non-negative integer
Default: No default

`multiplier` Multiplier that is applied to the projected value
Type: Real number
Default: 1.0
Lower bound: 0.0 (exclusive)
Value: A vector of length 1 (for a constant value for all years), or a vector of length years (for a specific value for each year)

9.11.4 Project of type Lognormal

`@project[label].type=Lognormal`. See Section 5.12.5 for more information.

`mean` The mean of the lognormal process
Type: Real number
Default: 0.0

`sigma` The standard deviation (sigma) of the lognormal sampling
Type: Real number
Default: No default
Lower bound: 0.0 (inclusive)

`multiplier` Multiplier that is applied to the projected value
Type: Real number
Default: 1.0
Lower bound: 0.0 (exclusive)
Value: A vector of length 1 (for a constant value for all years), or a vector of length years (for a specific value for each year)

9.11.5 Project of type Lognormal Empirical

`@project[label].type=Lognormal_Empirical`. See Section 5.12.6 for more information.

`mean` The mean of the Gaussian process
Type: Real number
Default: 0.0

`start_year` The start year of sampling
Type: Non-negative integer
Default: No default

`final_year` The final year of sampling

Type: Non-negative integer
Default: No default

`multiplier` Multiplier that is applied to the projected value
Type: Real number
Default: 1.0
Lower bound: 0.0 (exclusive)
Value: A vector of length 1 (for a constant value for all years), or a vector of length years (for a specific value for each year)

9.11.6 Project of type Harvest Strategy Constant Catch

`@Project[label].type=harvest_strategy_constant_catch`. See Section 5.12.8 for more information.

`catch` The catch to apply
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)

`alpha` The multiplier on the proportional change in biomass applied to the catch
Type: Real number
Default: 1.0
Lower bound: 0 (exclusive)

`min_delta` The minimum difference (proportion) in catch required before it is updated
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)

`max_delta` The maximum difference (proportion) in catch that can be applied
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)
Note: (
Use max_delta = 0 to have no maximum)

`update_frequency_years` The number of years between updates
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`biomass_index` The biomass index for calculating the changes in catch
Type: String
Default: No default
Value: A valid derived quantity label

`biomass_index_lag_years` The lag (years) of the derived_quantity that is used for the

calculation of the catch
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`current_catch` The current catch to apply at the start of the projections (applied until first_year)
Type: Real number
Default: 0
Lower bound: 0 (inclusive)

`multiplier` Multiplier that is applied to the projected value
Type: Vector of real numbers
Default: 1.0
Lower bound: 0 (exclusive)

`first_year` The first year in which to consider an update using the harvest strategy rule
Type: Non-negative integer
Default: 0

9.11.7 Project of type Harvest Strategy Constant U

`@Project[label].type=Harvest_Strategy_Constant_U`. See Section 5.12.9 for more information.

`u` The exploitation rate to apply (U)
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)
Upper bound: 1 (inclusive)

`min_delta` The minimum difference (proportion) in catch required before it is updated
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)

`max_delta` The maximum difference (proportion) in catch that can be applied
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)
Note: (
Use max_delta = 0 to have no maximum)

`update_frequency_years` The number of years between updates
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`biomass_index` The biomass index for calculating the changes in U

Type: String
Default: No default
Value: A valid derived quantity label

`biomass_index_scalar` A multiplicative scalar on the biomass index used for calculating the changes in U
Type: Real number
Default: 1.0
Lower bound: 0 (inclusive)

`biomass_index_lag_years` The lag (years) of the derived_quantity that is used for the calculation of the catch
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`current_catch` The current catch to apply at the start of the projections (applied until first_year)
Type: Real number
Default: 0
Lower bound: 0 (inclusive)

`multiplier` Multiplier that is applied to the projected value
Type: Vector of real numbers
Default: 1.0
Lower bound: 0 (exclusive)

`first_year` The first year in which to consider an update using the harvest strategy rule
Type: Non-negative integer
Default: 0

9.11.8 Project of type Harvest Strategy Ramp U

`@Project[label].type=harvest_strategy_ramp_u`. See Section 5.12.10 for more information.

`u` The exploitation rates to apply
Type: Vector of real numbers
Default: 0.0
Lower bound: 0 (inclusive)

`reference_points` The reference points for each exploitation rate
Type: Vector of real numbers
Default: 0.0
Lower bound: 0 (inclusive)

`reference_index` The biomass index for reference points (i.e., the derived quantity label for

the calculation of reference points)
Type: String
Default: No default
Value: A valid derived quantity label

`bias_adjustment` Whether to apply the bias adjustment for the lognormal derived quantity in the calculation of the ratio of the initial and current reference index
Type: boolean
Default: False

`min_delta` The minimum difference (proportion) in catch required before it is updated
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)

`max_delta` The maximum difference (proportion) in catch that can be applied
Type: Real number
Default: 0.0
Lower bound: 0 (inclusive)
Note: (
Use max_delta = 0 to have no maximum)

`update_frequency_years` The number of years between updates
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`biomass_index` The biomass index for calculating the changes in U
Type: String
Default: No default
Value: A valid derived quantity label

`biomass_index_scalar` A multiplicative scalar on the biomass index used for calculating the changes in U
Type: Real number
Default: 1.0
Lower bound: 0 (inclusive)

`biomass_index_lag_years` The lag (years) of the derived_quantity that is used for the calculation of the catch
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`current_catch` The current catch to apply at the start of the projections (applied until first_year)
Type: Real number
Default: 0
Lower bound: 0 (inclusive)

`multiplier` Multiplier that is applied to the projected value
Type: Vector of real numbers
Default: 1.0
Lower bound: 0 (exclusive)

`first_year` The first year in which to consider an update using the harvest strategy rule
Type: Non-negative integer
Default: 0

`b0_initialisation_phase` The phase that defines the initial biomass (e.g., B0)
Type: Non-negative integer
Default: The first initialisation phase

10 Estimation command and subcommand syntax

The description of methods for the estimation section is given in Section 6.

In the following section, the sub-section headers use a notation of the form "**@observation[label].type=abundance**" which, in this case, represents the input command fragment

```
@observation label # where label is a unique label for that observation
type=abundance
...
```

The specific subcommands for a command are given within each command.

10.1 Estimation methods

`@estimate` *`label`* Define an object of type *Estimate*. See Section 6 for more information.

`label` The label of the estimate
Type: String
Default: No default

`type` The type of prior for the estimate
Type: String
Default: No default

`parameter` The name of the parameter to estimate
Type: String
Default: No default

`lower_bound` The lower bound for the parameter
Type: Real number (estimable)
Default: No default

`upper_bound` The upper bound for the parameter
Type: Real number (estimable)
Default: No default

`same` List of other parameters that are constrained to have the same value as this parameter
Type: Vector of strings
Default: No default

`estimation_phase` The first estimation phase to allow this to be estimated
Type: Non-negative integer
Default: 1
Value: Phases must be number sequentially and start at one

`mcmc_fixed` Indicates if this parameter is estimated at the point estimate but fixed during MCMC estimation run
Type: Boolean
Default: false

10.1.1 Estimate of type Uniform

`@estimate[label].type=Uniform`. See Section 6.7.1 for more information.

The Uniform type has no additional subcommands.

10.1.2 Estimate of type Uniform-Log

`@estimate[label].type=Uniform_Log`. See Section 6.7.2 for more information.

The Uniform_Log type has no additional subcommands.

10.1.3 Estimate of prior type Normal

`@estimate[label].type=Normal`. See Section 6.7.3 for more information.

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

`cv` The normal standard deviation (sigma) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

10.1.4 Estimate of prior type Normal By Stdev

`@estimate[label].type=Normal_By_Stdev`. See Section 6.7.4 for more information.

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

`sigma` The normal standard deviation (sigma) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`lognormal_transformation` Add a Jacobian if the derived outcome of the estimate is assumed to be lognormal, e.g., used for recruitment deviations in the recruitment process. See the User Manual for more information
Type: Boolean
Default: false

10.1.5 Estimate of prior type Lognormal

`@estimate[label].type=Lognormal`. See Section 6.7.5 for more information.

`mu` The lognormal prior mean (mu) parameter

Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`cv` The lognormal variance (cv) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

10.1.6 Estimate of prior type Normal-Log

`@estimate[label].type=Normal_Log`. See Section 6.7.6 for more information.

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

`sigma` The normal-log prior standard deviation (sigma) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

10.1.7 Estimate of prior type Beta

`@estimate[label].type=Beta`. See Section 6.7.7 for more information.

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

`sigma` Beta prior standard deviation (sigma) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a` Beta prior lower bound of the range (A) parameter
Type: Real number (estimable)
Default: No default

`b` Beta prior upper bound of the range (B) parameter
Type: Real number (estimable)
Default: No default

10.1.8 Estimate of prior type Student's t

`@estimate[label].type=students_t`. See Section 6.7.8 for more information.

`mu` The Student's t prior location (mu) parameter

Type: Real number (estimable)
Default: No default

`sigma` The Student's t scale (sigma) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`df` The Student's t degrees of freedom (df) parameter
Type: Real number (constant)
Default: No default
Lower bound: 0.0 (exclusive)

10.2 Point estimation

`@minimiser` *`label`* Define an object of type *Minimiser*. See Section 6.4 for more information.

`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

10.2.1 Minimiser of type ADOLC

`@minimiser[label].type=ADOLC`. See Section 6.4.5 for more information.

`iterations` The maximum number of iterations
Type: Integer
Default: 1000
Lower bound: 1 (inclusive)

`evaluations` The maximum number of evaluations
Type: Integer
Default: 4000
Lower bound: 1 (inclusive)

`tolerance` The tolerance of the gradient for convergence
Type: Real number
Default: 1e-5
Lower bound: 0.0 (exclusive)

`step_size` The minimum step-size before minimisation fails
Type: Real number
Default: 1e-7
Lower bound: 0.0 (exclusive)

`parameter_transformation` The choice of parametrisation used to scale the parameters for ADOLC
Type: string
Default: sin_transformation
Value: Either `sin_transform` or `tan_transform`. See 6.4.5 for more information

10.2.2 Minimiser of type Betadiff

`@minimiser[label].type=Betadiff`. See Section 6.4.4 for more information.

`iterations` The maximum number of iterations
Type: Integer
Default: 1000
Lower bound: 1 (inclusive)

`evaluations` The maximum number of evaluations
Type: Integer
Default: 4000
Lower bound: 1 (inclusive)

`tolerance` The tolerance of the gradient for convergence
Type: Real number
Default: 1e-5
Lower bound: 0.0 (exclusive)

10.2.3 Minimiser of type DESolver

`@minimiser[label].type=de_solver`. See Section 6.4.3 for more information.

`population_size` The number of candidate solutions to have in the population
Type: Non-negative integer
Default: 25
Lower bound: 1 (inclusive)

`crossover_probability` The minimiser's crossover probability

Type: Real number
Default: 0.9
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

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

`max_generations` The maximum number of iterations to run
Type: 1000
Default: No default

`tolerance` The total variance between the population and best candidate before acceptance
Type: Real number
Default: 1e-5
Lower bound: 0.0 (exclusive)

10.2.4 Minimiser of type Deltadiff

`@minimiser[label].type=Deltadiff`. See Section 6.4.2 for more information.

`iterations` Maximum number of iterations
Type: Integer
Default: 1000
Lower bound: 1 (inclusive)

`evaluations` Maximum number of evaluations
Type: Integer
Default: 4000
Lower bound: 1 (inclusive)

`tolerance` Tolerance of the gradient for convergence
Type: Real number
Default: 1e-5
Lower bound: 0 (exclusive)

`step_size` Minimum Step-size before minimisation fails
Type: Real number
Default: 1e-7
Lower bound: 0 (exclusive)

10.2.5 Minimiser of type Numerical Differences

`@minimiser[label].type=Numerical_Differences`. See Section 6.4.1 for more information.

`iterations` The maximum number of iterations

Type: Integer
Default: 1000
Lower bound: 1 (inclusive)

`evaluations` The maximum number of evaluations
Type: Integer
Default: 4000
Lower bound: 1 (inclusive)

`tolerance` The tolerance of the gradient for convergence
Type: Real number
Default: 1e-5
Lower bound: 0.0 (exclusive)

`step_size` The minimum step-size before minimisation fails
Type: Real number
Default: 1e-7
Lower bound: 0.0 (exclusive)

10.3 Markov chain Monte Carlo (MCMC)

`@mcmc` *`label`* Define an object of type *MCMC*. See Section 6.6 for more information.

`label` The label of the MCMC
Type: String
Default: No default

`type` The MCMC method
Type: String
Default: No default

`length` The number of iterations for the MCMC (including the burn in period)
Type: Non-negative integer
Default: No default
Lower bound: 1 (inclusive)

`burn_in` The number of iterations for the burn_in period of the MCMC
Type: Non-negative integer
Default: 0
Lower bound: 0 (inclusive)

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

`step_size` Initial step-size (as a multiplier of the approximate covariance matrix)

Type: Real number
Default: The default is $2.4d^{-0.5}$
Lower bound: 0 (inclusive)
Note: If the value is set to zero or the subcommand is omitted, the default value is used instead

`start` The covariance multiplier for the starting point of the MCMC
Type: Real number
Default: 0.0
Lower bound: 0.0 (inclusive)
Value: If zero, then the MCMC starts at the point estimate (i.e., the MPD). Otherwise a random (multivariate normal) jump from the point estimate with `start` used as the standard deviation multiplier

`adjust_parameters_at_bounds` Adjust the start point for parameters at bounds
Type: Boolean
Default: false
Value: If true, then the MCMC will adjust the start point of any parameters at a bound to a random uniform location between the lower and upper bound

`keep` The spacing between recorded values in the MCMC
Type: Non-negative integer
Default: 1
Lower bound: 1 (inclusive)

`max_correlation` The maximum absolute correlation in the covariance matrix of the proposal distribution
Type: Real number
Default: 0.8
Lower bound: 0.0 (exclusive)
Upper bound: 1.0 (inclusive)

`covariance_adjustment_method` The method for adjusting small variances in the covariance proposal matrix
Type: String
Default: correlation
Value: Either covariance, correlation, or none

`correlation_adjustment_diff` The minimum non-zero variance times the range of the bounds in the covariance matrix of the proposal distribution
Type: Real number
Default: 0.0001
Lower bound: 0.0 (exclusive)

`proposal_distribution` The shape of the proposal distribution (either the t or the normal distribution)
Type: String
Default: t
Value: Either t or normal

`df` The degrees of freedom of the multivariate t proposal distribution
Type: Non-negative integer
Default: 4
Lower bound: 1

`adapt_stepsize_at` The iteration numbers in which to check and resize the MCMC step-size
Type: Vector of non-negative integers
Default: true
Lower bound: 0 (inclusive)
Value: If zero, then no step-size adaption is applied. Otherwise must be a positive integer less than less than the burn_in

`adapt_stepsize_method` The method to use to adapt the step-size
Type: String
Default: ratio
Value: Either `double_half` or `ratio`

`adapt_covariance_matrix_at` The iteration number in which to adapt the covariance matrix
Type: Non-negative integer
Default: 0
Lower bound: 0 (inclusive)
Value: If zero, then no covariance matrix adaption is applied. Otherwise must be a positive integer that is less than the burn_in

10.3.1 MCMC of type Hamiltonian Monte Carlo

`@mcmc[label].type=Hamiltonian`.

`leapfrog_steps` Number of leapfrog steps
Type: Non-negative integer
Default: 1
Lower bound: 0 (exclusive)

`leapfrog_delta` Amount to leapfrog per step
Type: Real number
Default: 1e-7
Lower bound: 0 (exclusive)

`gradient_step_size` Step-size to use when calculating gradient
Type: Real number
Default: 1e-7
Lower bound: 1e-13 (inclusive)

10.3.2 MCMC of type Random Walk Metropolis Hastings

`@mcmc[label].type=Random_Walk`. See Section 6.6 for more information.

The Random_Walk type has no additional subcommands.

10.4 Posterior profiles

@profile *label* Define an object of type *Profile*. See Section 6.5 for more information.

`label` The label of the profile
Type: String
Default: No default

`steps` The number of steps between the lower and upper bound
Type: Non-negative integer
Default: No default
Value: A positive integer ≥ 2

`lower_bound` The lower value of the range
Type: Real number (estimable)
Default: No default

`upper_bound` The upper value of the range
Type: Real number (estimable)
Default: No default

`parameter` The free parameter to profile
Type: String
Default: No default

`same` A free parameter that is constrained to have the same value as the parameter being profiled
Type: String
Default: No default

`transformation` The transformation to apply to the upper and lower bounds
Type: String
Default: 'none'
Value: 'none', 'log', 'square_root', and 'inverse'
Note: This specifies that the upper and lower bounds should be transformed from natural space into the transformed space before being evaluated and used for the profile

10.5 Catchability constants

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

`label` Label of the catchability
Type: String
Default: No default

`type` The type of catchability
Type: String
Default: No default

10.5.1 Catchability of type Free

`@catchability[label].type=Free`.

`q` The value of the catchability
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)

10.5.2 Catchability of type Nuisance

`@catchability[label].type=Nuisance`.

`lower_bound` The upper bound for nuisance catchability
Type: Real number (estimable)
Default: No default

`upper_bound` The lower bound for nuisance catchability
Type: Real number (estimable)
Default: No default

`q` The value of the catchability
Type: Addressable
Default: No default

10.6 Penalties

`@penalty` *`label`* Define an object of type *Penalty*. See Section 6.8 for more information.

`label` The label of the penalty
Type: String
Default: No default

`type` The type of penalty
Type: String
Default: No default

10.6.1 Penalty of type Process

`@penalty[label].type=Process`. See Section 6.8 for more information.

`multiplier` The penalty multiplier
Type: Real number (estimable)
Default: 1.0

`log_scale` Indicates if the sums of squares will be calculated on the log scale
Type: Boolean
Default: false

10.7 Additional priors

`@additional_prior` *`label`* Define an object of type *Additional_Prior*. See Section 6.9 for more information.

`label` The label for the additional prior
Type: String
Default: No default

`type` The additional prior type
Type: String
Default: No default

10.7.1 Additional Prior of type Beta

`@additional_prior[label].type=Beta`. See Section 6.9.7 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

`mu` Beta distribution mean μ parameter
Type: Real number (estimable)
Default: No default

`sigma` Beta distribution variance σ parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`a` Beta distribution lower bound, of the range A parameter
Type: Real number
Default: No default

`b` Beta distribution upper bound of the range B parameter
Type: Real number
Default: No default

10.7.2 Additional Prior of type Element Difference

`@additional_prior[label].type=Element_Difference`. See Section 6.9.6 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

`second_parameter` The name of the second parameter for comparing
Type: String
Default: No default

`multiplier` Multiply the penalty by this factor
Type: Real number
Default: 1

10.7.3 Additional Prior of type Log Normal

`@additional_prior[label].type=Log_Normal`. See Section 6.9.3 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

`mu` The lognormal prior mean (mu) parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

`cv` The lognormal CV parameter
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

10.7.4 Additional Prior of type Uniform

`@additional_prior[label].type=Uniform`. See Section 6.9.5 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

10.7.5 Additional Prior of type Uniform-Log

`@additional_prior[label].type=Uniform_Log`. See Section 6.9.4 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

10.7.6 Additional Prior of type Vector Average

`@additional_prior[label].type=Vector_Average`. See Section 6.9.2 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

`method` Which calculation method to use: k, l, or m
Type: String
Default: k

`k` The k value to use in the calculation
Type: Real number
Default: No default

`multiplier` Multiplier for the penalty amount
Type: Real number
Default: 1

10.7.7 Additional Prior of type Vector Smoothing

`@additional_prior[label].type=Vector_Smoothing`. See Section 6.9.1 for more information.

`parameter` The name of the parameter for the additional prior
Type: String
Default: No default

`log_scale` Should the sums of squares be calculated on the log scale?
Type: Boolean
Default: false

`multiplier` Multiply the penalty by this factor
Type: Real number
Default: 1

`lower_bound` The first element to apply the penalty to in the vector
Type: Non-negative integer
Default: 0

`upper_bound` The last element to apply the penalty to in the vector
Type: Non-negative integer
Default: 0

`r` The rth difference that the penalty is applied to
Type: Non-negative integer
Default: 2

10.7.8 Additional Prior of type Sum

`@additional_prior[label].type=Sum`. See Section 6.9.8 for more information.

`parameters` The names of the parameters for summing

Type: Vector of strings
Default: No default

`distribution` The additional prior distribution to apply
Type: String
Default: lognormal
Value: (
Either normal or lognormal)

`mu` Mean of the distribution
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (exclusive)

`cv` cv of the distribution
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (exclusive)

10.8 Parameter transformations

`@parameter_transformation` *`label`* Define an object of type *parameter_transformation*. See Section 6.10 for more information.

`label` Label for the transformation block
Type: String
Default: No default

`type` The type of transformation
Type: String
Default: No default

`prior_applies_to_restored_parameters` If the prior applies to the parameters (true) with jacobian (if it exists) or prior applies to transformed_parameter (false) with no jacobian
Type: bool
Default: false

`parameters` The label of the parameters used in the transformation
Type: String
Default: No default

10.8.1 Parameter transformation of type Log

`@parameter_transformation[label].type=log`.

The addressable parameter for this transformation is `log_parameter`. See Section 6.10.2, paragraph 1.

10.8.2 Parameter transformation of type Logistic

`@parameter_transformation[label].type=logistic`.
The addressable parameter for this transformation is `logistic_parameter`. See Section 6.10.2, paragraph 7.

`lower_bound` Lower bound for the transformation
Type: Numeric
Default: No default

`upper_bound` Upper bound for the transformation
Type: Numeric
Default: No default

10.8.3 Parameter transformation of type Inverse

`@parameter_transformation[label].type=inverse`.

The addressable parameter for this transformation is `inverse_parameter`. See Section 6.10.2, paragraph 2.

10.8.4 Parameter transformation of type Difference

`@parameter_transformation[label].type=difference_parameter`.

The addressable parameter for this transformation is `difference_parameter`. See Section 6.10.2, paragraph 3.

10.8.5 Parameter transformation of type Average Difference

`@parameter_transformation[label].type=average_difference`.

The addressable parameters for this transformation are `average_parameter` and `difference_parameter`. See Section 6.10.2, paragraph 4.

10.8.6 Parameter transformation of type log sum

`@parameter_transformation[label].type=log_sum`.

The addressable parameters for this transformation are `log_total_parameter` and `total_proportion_parameter`. See Section 6.10.2, paragraph 5.

10.8.7 Parameter transformation of type orthogonal

`@parameter_transformation[label].type=orthogonal`.

The addressable parameters for this transformation are `product_parameter` and `quotient_parameter`. See Section 6.10.2, paragraph 6.

10.8.8 parameter transformation of type sum to one

`@parameter_transformation[label].type=sum_to_one`.

The addressable parameter for this transformation is `product_parameter`. See Section 6.10.2, paragraph 8.

10.8.9 parameter transformation of type simplex

`@parameter_transformation[label].type=simplex`.

`sum_to_one` Apply the sum_to_one constraint
Type: bool
Default: true
Value: If true, the parameter vector in natural space will sum to one, otherwise it will sum to the value of the length(parameter), i.e., defines the vector to have average one

The addressable parameter for this transformation is `simplex`. See Section 6.10.2, paragraph 9.

10.8.10 Parameter transformation of type square root

`@parameter_transformation[label].type=sqrt`.

The addressable parameter for this transformation is `sqrt_parameter`. See Section 6.10.2, paragraph 10.

11 Observation command and subcommand syntax

The description of methods for the observation section is given in Section 7.

In the following section, the sub-section headers use a notation of the form "**@observation[label].type=abundance**" which, in this case, represents the input command fragment

```
@observation label # where label is a unique label for that observation
type=abundance
...
```

The specific subcommands for a command are given within each command.

11.1 Observation types

The description of the observations is given in Section 7. 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*. See Section 7 for more information.

`label` The label of the observation
Type: String
Default: No default

`type` The type of observation
Type: String
Default: No default

`likelihood` The type of likelihood to use
Type: String
Default: No default

`categories` The category labels to use
Type: Vector of strings
Default: true

`delta` The robustification value (delta) for the likelihood
Type: Real number (estimable)
Default: 1e-11
Lower bound: 0.0 (inclusive)

`simulation_likelihood` The simulation likelihood to use
Type: String
Default: No default

`likelihood_multiplier` The likelihood multiplier
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (inclusive)

`error_value_multiplier` The error value multiplier for likelihood
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (inclusive)

11.1.1 Observation of type Abundance

`@observation[label].type=Abundance`. See Section 7.1.1 for more information.

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

`catchability` The label of the catchability coefficient (q)
Type: String
Default: No default

`selectivities` The labels of the selectivities
Type: Vector of strings
Default: true

`process_error` The process error
Type: Real number (estimable)
Default: 0.0
Lower bound: 0.0 (inclusive)

`years` The years for which there are observations
Type: Vector of non-negative integers
Default: No default

`table obs` The table of data specifying the observed and error values
Type: Data table with label = obs
Default: No default
Value: A $n*3$ matrix, where $n =$ the years and a column for year, observation and error value. See below for example.
Note: example below

```
table obs
1993 238.2 0.12
1994 170 0.16
1995 216.2 0.18
```

```
2004 46.9 0.20
end_table
```

11.1.2 Observation of type Biomass

`@observation[label].type=Biomass`. See Section 7.1.1 for more information.

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

`catchability` The label of the catchability coefficient (q)
Type: String
Default: No default

`selectivities` The labels of the selectivities
Type: Vector of strings
Default: true

`process_error` The process error
Type: Real number (estimable)
Default: 0.0
Lower bound: 0.0 (inclusive)

`age_weight_labels` The labels for the `@`*age_weight* block which corresponds to each category, to use the weight calculation method for biomass calculations)
Type: Vector of strings
Default: No default

`years` The years of the observed values
Type: Vector of non-negative integers
Default: No default

`table obs` The table of data specifying the observed and error values
Type: Data table with label = obs
Default: No default
Value: A $n * 3$ matrix, where $n =$ the years and a column for year, observation and error value. See below for example.
Note: example below

```
table obs
# year observation error_value
1993 238.2 0.12
1994 170 0.16
1995 216.2 0.18
2004 46.9 0.20
end_table
```

11.1.3 Observation of type Process Removals By Length

`@observation[label].type=Process_Removals_By_Length`. See Section 7.1.3 for more information.

`length_bins` Bespoke length bins for the observation. They need to be a subset of the `@model` length bins.
Type: Non-negative integer
Default: Model length bins

`sum_to_one` Scale the year (row) observed values by the total, so they sum to 1
Type: Boolean
Default: false

`simulated_data_sum_to_one` Whether simulated data are discrete or scaled by totals to be proportions for each year
Type: Boolean
Default: true

`plus_group` Is the maximum length bin a plus group
Type: Boolean
Default: true

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

`years` The years for which there are observations
Type: Vector of non-negative integers
Default: No default

`process_errors` The label of process error to use
Type: Vector of real numbers (estimable)
Default: true

`method_of_removal` The label of the observed method of removals
Type: Vector of strings
Default: No default

`mortality_instantaneous_process` The label of the mortality instantaneous process for the observation
Type: String
Default: No default

`table obs` The table of data specifying the observed values

Type: Data table with label = obs
Default: No default
Value: A $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table obs
1993 0.1 0.2 0.3
1994 0.1 0.2 0.3
end_table
```

`table error_values` The table of data specifying the error values
Type: Data table with label = error_values
Default: No default
Value: Can be specified two ways either as a $n \times 1$ matrix with an error value for each year. Or a $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table error_values
1993 234
1994 343
end_table
```

11.1.4 Observation of type Proportions At Length

`@observation[label].type=Proportions_At_Length`. See Section 7.1.1 for more information.

`length_bins` Bespoke length bins for the observation. They need to be a subset of the `@model`length_bins.
Type: Non-negative integer
Default: Model length bins

`plus_group` Is the maximum length bin a plus group
Type: Boolean
Default: true

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

`years` The years of the observed values
Type: Vector of non-negative integers
Default: No default

`selectivities` The labels of the selectivities
Type: Vector of strings
Default: true

`process_errors` The process error
Type: Vector of real numbers (estimable)
Default: true

`sum_to_one` Scale the year (row) observed values by the total, so they sum to 1
Type: Boolean
Default: false

`simulated_data_sum_to_one` Whether simulated data is discrete or scaled by totals to be proportions for each year
Type: Boolean
Default: true

`table obs` The table of data specifying the observed values
Type: Data table with label = obs
Default: No default
Value: A $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table obs
1993 0.1 0.2 0.3
1994 0.1 0.2 0.3
end_table
```

`table error_values` The table of data specifying the error values
Type: Data table with label = error_values
Default: No default
Value: Can be specified two ways either as a $n \times 1$ matrix with an error value for each year. Or a $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table error_values
1993 234
1994 343
end_table
```

11.1.5 Observation of type Proportions By Category

`@observation[label].type=Proportions_By_Category`. See Section 7.1.1 for more information.

`length_bins` Bespoke length bins for the observation. They need to be a subset of the `@model`length_bins.
Type: Non-negative integer
Default: Model length bins

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

Type: String
Default: No default

`plus_group` Use the age plus group?
Type: Boolean
Default: true

`years` The years for which there are observations
Type: Vector of non-negative integers
Default: No default

`selectivities` The labels of the selectivities
Type: Vector of strings
Default: true

`total_categories2` Categories in the denominator
Type: Vector of strings
Default: No default

`total_selectivities` Selectivities to apply to the total categories
Type: Vector of strings
Default: No default

11.1.6 Observation of type Tag Recapture By Length

`@observation[label].type=Tag_Recapture_By_Length`. See Section 7.1.1 for more information.

`years` The years for which there are observations
Type: Vector of non-negative integers
Default: No default

`time_step` The time step to execute in
Type: String
Default: No default

`length_bins` The length bins
Type: Vector of real numbers (estimable)
Default: true

`selectivities` The labels of the selectivities used for untagged categories
Type: Vector of strings
Default: true

`tagged_selectivities` The labels of the tag category selectivities
Type: Vector of strings
Default: No default

`detection` The probability of detecting a recaptured individual
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`dispersion` The overdispersion parameter (phi)
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (inclusive)

`time_step_proportion` The proportion through the mortality block of the time step when the observation is evaluated
Type: Real number (estimable)
Default: 0.5
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`table recaptured` The table of data specifying the recaptures
Type: Data table with label = recaptured
Default: No default
Value: A $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table recaptured
1993 1 32 25
1994 3 4 43
end_table
```

`table scanned` The table of data specifying the scanned fish
Type: Data table with label = scanned
Default: No default
Value: A $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table scanned
1993 1 32 25
1994 3 4 43
end_table
```

11.1.7 Observation of type Tag Recapture By Length For growth

`@observation[label].type=tag_recapture_by_length_for_growth`. See Section 7.1.1 for more information.

`years` The years for which there are observations

Type: Vector of non-negative integers
Default: No default

`time_step` The time step to execute in
Type: String
Default: No default

`length_bins` The length bins
Type: Vector of real numbers (estimable)
Default: true

`selectivities` The labels of the selectivities used for untagged categories
Type: Vector of strings
Default: true

`tagged_selectivities` The labels of the tag category selectivities
Type: Vector of strings
Default: No default

`detection` The probability of detecting a recaptured individual
Type: Real number (estimable)
Default: No default
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`dispersion` The overdispersion parameter (phi)
Type: Real number (estimable)
Default: 1.0
Lower bound: 0.0 (inclusive)

`time_step_proportion` The proportion through the mortality block of the time step when the observation is evaluated
Type: Real number (estimable)
Default: 0.5
Lower bound: 0.0 (inclusive)
Upper bound: 1.0 (inclusive)

`table recaptured` The table of data specifying the recaptures
Type: Data table with label = recaptured
Default: No default
Value: A $n \times m$ matrix, where $n =$ the years and m is categories $\times$ length bins. See below for example.
Note: example below

```
table recaptured
1993 1 32 25
1994 3 4 43
```

```
end_table
```

11.2 Likelihoods

`@likelihood` *`label`* Define an object of type *Likelihood*. See Section 7.2 for more information.

11.2.1 Likelihood of type Binomial

`@likelihood[label].type=Binomial`.

The Binomial type has no additional subcommands.

11.2.2 Likelihood of type Binomial Approx

`@likelihood[label].type=Binomial_Approx`.

The Binomial_Approx type has no additional subcommands.

11.2.3 Likelihood of type Dirichlet

`@likelihood[label].type=Dirichlet`.

The Dirichlet type has no additional subcommands.

11.2.4 Likelihood of type Dirichlet-Multinomial

`@likelihood[label].type=Dirichlet_Multinomial`.

The Dirichlet-Multinomial type has no additional subcommands.

11.2.5 Likelihood of type Log Normal

`@likelihood[label].type=Log_Normal`.

The Log_Normal type has no additional subcommands.

11.2.6 Likelihood of type Log Normal With Q

`@likelihood[label].type=Log_Normal_With_Q`.

The Log_Normal_With_Q type has no additional subcommands.

11.2.7 Likelihood of type Multinomial

`@likelihood[label].type=Multinomial`.

The Multinomial type has no additional subcommands.

11.2.8 Likelihood of type Normal

`@likelihood[label].type=Normal`.

The Normal type has no additional subcommands.

11.2.9 Likelihood of type Pseudo

`@likelihood[label].type=none`.

The Pseudo type has no additional subcommands.

11.2.10 Likelihood of type Bernoulli

`@likelihood[label].type=bernoulli`.

12 Report command and subcommand syntax

The description of each report is given in Section 8.

12.1 Report commands and subcommands

`@report` *`label`* Define an object of type *Report*. See Section 8 for more information.

`label` The report label
Type: String
Default: No default

`type` The report type
Type: String
Default: No default

`file_name` The file name. If not supplied, then output is directed to standard out
Type: String
Default: No default

`write_mode` Specify if any previous file with the same name should be overwritten, appended to, or is generated using a sequential suffix
Type: String
Default: overwrite
Value: valid options are `append`, `overwrite`, `incremental_suffix`

`format` Report output format
Type: String
Default: r
Value: Either **R** for formatting for reading into **R** or `none` for no formatting

12.1.1 Report of type Default

`@report[label].type=Default`. See Section 8.3 for more information.

`catchabilities` Report catchabilities
Type: Boolean
Default: false
Note: Reports all valid catchabilities

`derived_quantities` Report derived quantities
Type: Boolean
Default: false
Note: Reports all valid derived quantities

`observations` Report observations
Type: Boolean
Default: false
Note: Reports all valid observations

`processes` Report processes
Type: Boolean
Default: false
Note: Reports all valid processes

`projects` Report projects
Type: Boolean
Default: false
Note: Reports all valid projections

`selectivities` Report selectivities
Type: Boolean
Default: false
Note: Reports all valid selectivities

`time_varying` Report time-varying parameters
Type: Boolean
Default: false
Note: Reports all valid time-varying parameters

`parameter_transformations` Report all parameter transformations
Type: Boolean
Default: false
Note: Reports all valid parameter transformations

12.1.2 Report of type Addressable

`@report[label].type=Addressable`. See Section 8.8 for more information.

`parameter` The addressable parameter name
Type: String
Default: No default

`years` Define the years that the report is generated for
Type: Vector of non-negative integers
Default: No default

`time_step` Defines the time-step that the report applies to
Type: String
Default: No default
Value: A valid time step label

12.1.3 Report of type Growth Increment model

`@report[label].type=growth_increment`. See Section 8.9 for more information.

`time_step` The time step label

Type: String
Default: No default
Value: A valid time step label

`years` The years for the report
Type: Vector of non-negative integers
Default: All years

`growth_increment` The growth-increment label
Type: String
Default: No default

12.1.4 Report of type Catchability

`@report[label].type=Catchability`. See Section 8.21 for more information.

`catchability` The catchability label
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the category label

12.1.5 Report of type Correlation Matrix

`@report[label].type=Correlation_Matrix`. See Section 8.19 for more information.

The Correlation_Matrix report has no additional subcommands.

12.1.6 Report of type Covariance Matrix

`@report[label].type=Covariance_Matrix`. See Section 8.18 for more information.

The Covariance_Matrix type has no additional subcommands.

12.1.7 Report of type Derived Quantity

`@report[label].type=Derived_Quantity`. See Section 8.12 for more information.

`derived_quantity` The derived quantity label
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the derived quantity label

12.1.8 Report of type Equation Test

`@report[label].type=Equation_Test`. See Section 5.11 for more information.

`equation` The equation to do a test run of
Type: Vector of strings
Default: No default

12.1.9 Report of type Estimate Summary

`@report[label].type=Estimate_Summary`. See Section 8.13 for more information.

Value: A summary of the estimated (free parameters)

The Estimate_Summary type has no additional subcommands.

12.1.10 Report of type Estimate Value

`@report[label].type=Estimate_Value`. See Section 8.14 for more information.

Value: The free parameters and their values, in a format suitable for use with `-i`

The Estimate_Value report has no additional subcommands.

12.1.11 Report of type Estimation Result

`@report[label].type=Estimation_Result`. See Section 8.16 for more information.

Value: A summary of the results of the minimisation

The Estimation_Result report has no additional subcommands.

12.1.12 Report of type Hessian Matrix

`@report[label].type=Hessian_Matrix`. See Section 8.20 for more information.

The Hessian_Matrix report has no additional subcommands.

12.1.13 Report of type Initialisation

`@report[label].type=Initialisation`. See Section 8.4 for more information.

The Initialisation report has no additional subcommands.

12.1.14 Report of type Initialisation_Partition

`@report[label].type=Initialisation_Partition`. See Section 8.5 for more information.

12.1.15 Report of type MCMC Covariance

`@report[label].type=MCMC_Covariance`. See Section 6.6 for more information.

Value: This will output the covariance matrices (the initial covariance matrix and the covariance matrix if adapted) used for the MCMC chain.

The MCMC_Covariance report has no additional subcommands.

12.1.16 Report of type MCMC Objective

`@report[label].type=MCMC_Objective`. See Section 8.29 for more information.

The MCMC_Objective report has no additional subcommands.

`file_name` The file name. If not supplied the default filename is used

Type: string

Default: objectives

write_mode Has a different default to the rest of the reports.
Type: String
Default: incremental_suffix
Value: valid options are append, overwrite, incremental_suffix

12.1.17 Report of type MCMC Sample

@report[label].type=MCMC_Sample. See Section 8.28 for more information.

file_name The file name. If not supplied the default filename is used
Type: string
Default: samples

write_mode Has a different default to the rest of the reports.
Type: String
Default: incremental_suffix
Value: valid options are append, overwrite, incremental_suffix

The MCMC_Sample report has no additional subcommands.

12.1.18 Report of type Objective Function

@report[label].type=Objective_Function. See Section 8.17 for more information.

The Objective_Function type has no additional subcommands.

12.1.19 Report of type Observation

@report[label].type=Observation. See Section 8.22 for more information.

observation The observation label
Type: String
Default: No default

normalised_residuals Print Normalised Residuals?
Type: Boolean
Default: true
Note: Only generated if valid for associated likelihood

pearsons_residuals Print Pearsons Residuals?
Type: Boolean
Default: true
Note: Only generated if valid for associated likelihood

12.1.20 Report of type Output Parameters

@report[label].type=Output_Parameters. See Section 8.15 for more information.

The Output_Parameters report has no additional subcommands.

12.1.21 Report of Parameter transformations

`@report[label].type=parameter_transformation`. See Section 8.10 for more information.

`parameter_transformation` label of parameter transformation block
Type: String
Default: No default

12.1.22 Report of type Partition

`@report[label].type=Partition`. See Section 8.6 for more information.

`time_step` Time Step label
Type: String
Default: No default

`years` Years
Type: Vector of non-negative integers
Default: All years

12.1.23 Report of type Partition Biomass

`@report[label].type=Partition_Biomass`. See Section 8.7 for more information.

`time_step` The time step label
Type: String
Default: No default

`years` The years for the report
Type: Vector of non-negative integers
Default: All years

12.1.24 Report of type Process

`@report[label].type=Process`. See Section 8.11 for more information.

`process` The process label that is reported
Type: String
Default: No default
Value: A valid process label
Value: If not specified, then the label of the report is assumed to be the process label

12.1.25 Report of type Profile

`@report[label].type=Profile`. See Section 6.5 for more information.

12.1.26 Report of type Project

`@report[label].type=Project`. See Section 5.12 for more information.

`project` The project label that is reported

Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the projection label

12.1.27 Report of type Random Number Seed

`@report[label].type=Random_Number_Seed`. See Section 8.26 for more information.

The Random_Number_Seed type has no additional subcommands.

12.1.28 Report of type Selectivity

`@report[label].type=Selectivity`. See Section 8.24 for more information.

`selectivity` Selectivity name
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the selectivity label

`length_values` Length bins for reporting if a length-based selectivity in an age-based model
Type: Vector of real numbers
Default: If not specified and this is a length-based selectivity in an age-based model, then length bins specified for the model will be used
Note: It is a fatal error if this is a report for a length-based selectivity in an age-based model, but neither the length values or `@model.length_bins` were supplied

12.1.29 Report of type Selectivity By Year

`@report[label].type=selectivity_by_year`. See Section 8.25 for more information.

`selectivity` Selectivity name
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the selectivity label

`years` years to report the selectivity in
Type: String
Default: true
Value: If not specified will print for all years in of the model

`time_step` Time step label
Type: String
Default: No default
Note: This should not matter, but is required in order to identify the time step for each year when values are printed.

12.1.30 Report of type Simulated Observation

`@report[label].type=Simulated_Observation`. See Section 8.23 for more information.

`observation` The observation label
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the observation label

12.1.31 Report of type Time Varying

`@report[label].type=Time_Varying`. See Section 8.30 for more information.

`time_varying` The time varying label that is reported
Type: String
Default: No default
Value: If not specified, then the label of the report is assumed to be the time varying label

13 Including commands from other files

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

`file` The name of the external input configuration file to include

Type: string

Default: No default

Value: A valid input configuration file

Note: If `file_name` includes a space character, then it must be enclosed in quotes, for example `@include "my file.csl2"`. Also note that the `@include` does not denote the end of the previous command block as is the case for all other commands

14 Validating model values using asserts

Casal2 can validate or check certain addressables parameters as a part of testing and validation with the assert command. Asserts check the value of a specific addressables (for example, and observations, parameters, or the objective function). Asserts are one aspect of the internal tests Casal2 uses to ensure accuracy across versions and revisions (see Section 3.8)

14.1 Assert syntax

`@assert` *`label`* Define an object of type *Assert*. See Section 3.8 for more information.

`label` The label for the assert
Type: String
Default: No default

`type` The type of the assert
Type: String
Default: No default

14.1.1 Assert of type Addressable

`@assert[label].type=Addressable`.

`parameter` The addressable to check
Type: String
Default: No default

`years` The years to check addressable
Type: Vector of non-negative integers
Default: No default

`time_step` The time step to execute after
Type: String
Default: No default

`values` The values to check against the addressable
Type: Vector of non-negative integers
Default: No default

`tolerance` The tolerance of the difference test
Type: Real number
Default: 1e-5

`error_type` Report assert failures as either an error or warning
Type: String
Value: Either warning or error
Default: error

14.1.2 Assert of type Objective_Function

`@assert[label].type=Objective_Function`.

`value` Expected value of the objective function
Type: Real number (estimable)
Default: No default

`tolerance` The tolerance of the difference test
Type: Real number
Default: 1e-5

`error_type` Report assert failures as either an error or warning
Type: String
Value: Either warning or error
Default: error

14.1.3 Assert of type Partition

`@assert[label].type=Partition.category` Category to check population values for
Type: String
Default: No default

`values` Values expected in the partition
Type: Vector of real numbers (estimable)
Default: No default

`tolerance` The tolerance of the difference test
Type: Real number
Default: 1e-5

`error_type` Report assert failures as either an error or warning
Type: String
Value: Either warning or error
Default: error

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

Many users of Casal2 may be be starting with a functioning CASAL model. This section focuses on transitioning from CASAL to Casal2.

There are a range of reasons why Casal2 will output 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 assumed that the latter reasons should be rare, and the 'overall' behaviour when it comes to estimation will be the same between CASAL and Casal2.

Reasons why there may be different values reported between CASAL and Casal2 include:

- Report rounding. There are settings with respect to output in CASAL that set the number of significant figures for writing to files. So if values look truncated, this might be the reason.
- Priors on parameters that are turned off with `upper_bound` = `lower_bound`. In both CASAL and Casal2 the estimation of parameters can be turned off by setting the bounds equal. CASAL will evaluate the prior value and add this to the objective function. This contribution is a constant value so it will not affect parameter inference. It may however be confusing when comparing output between the two models.
- Default values. There are a lot of switches in these programs, and options like the `delta` in Casal2 or `r` parameter in CASAL for robustifying likelihoods can cause differences.
- The order of processes. CASAL has a predefined sequence in which it executes processes within a time step (i.e., ageing, recruitment, maturation, migration, growth, natural and fishing mortality, disease mortality, tag release events, tag shedding rate, and semelparous mortality), where as Casal2 is completely user defined.
- Length-based processes or observations. Casal2 has updated the cumulative normal distribution calculation (CASAL used the approximated no closed form solution) with better approximations.
- Compositional observations. CASAL will only normalise (scale by the total) if the sum of proportions for a year are greater than 1.01 or less than 0.99. Casal2 will re normalise the prooprtions for a year even if they sum to one. If the observations are within those bounds technically CASAL and Casal2 will have slightly different observations and will generate small differences in likelihoods.
- Tag penalties. CASAL applies a penalty to the sum of squares on total tagged fish in a 'tagging episode' from the model compared to observed number of tagged fish. Casal2 applies a penalty on the transition rate by length. If tags are applied in a length bin that does not have individuals, e.g., a model configuration which tags 2 individuals of length l when there are no individuals in that length bin will include a penalty.

Many of the flags and options in CASAL and Casal2 are the same or similar. The syntax section of this document (Section 9) provides more details about the Casal2 functionality and behaviour. Check that the programs produce the same results with a **range** of parameter values using the deterministic run command (`casal2 -r`), before doing an estimation run (`casal2 -e`).

The first outputs to check when comparing Casal2 and CASAL versions of the same model are the stock dynamics outputs, ignoring the fits to observations. That is, check the initial age structure, the SSB and YCS values and patterns, R0, B0, etc. If these outputs differ, then the fits to the observations will likely also be different.

There are a few linkages with certain stock dynamics outputs to check to determine if processes

are misspecified. Differences between the proportions in the initial age structure, assuming an equilibrium state, are due to M, natural mortality. Differences in the initial equilibrium recruitment value, R_0, are due to growth (`@age_length` or `@length_weight`). Many models estimate B_0 so that R_0 is a back calculation through the growth curve.

If the initial age structure is the same, next check the derived quantities such as the SSB values. Differences in these values are generally caused by how fishing and recruitment processes are specified. Check which *YCS* values are estimated or standardised, the definition and designation of selectivities, etc.

Once the stock dynamics outputs match, check the results with a few different sets of starting parameter values by using the `-i` command line option. Next, check the fits to the observation data by comparing the expected values. Assuming the observations in both models match, the differences in the objective function value come from the expected values and the likelihood configurations. This is where subcommands such as the robustification values and the default values may differ between CASAL and Casal2.

Once the stock dynamics outputs and the fits to the observation data are the same, do an estimation run (`casal2 -e`). If CASAL and Casal2 do not optimise to the same parameter values, then use the parameter values from CASAL and do a deterministic run with Casal2 using the CASAL estimated parameter values (`casal2 -r -i CASAL_mpd_pars.txt`). Then check the stock dynamics outputs and the fits to the observation data and determine where the differences in the parameter estimates and outputs are.

The next question is, how close do the parameter estimates, expected values, and objective function values have to be to say that the models are equivalent? This is an ongoing topic of discussion. Previously, subjective qualitative measures have been used to decide whether the models are equivalent. A recorded comparison for the hake stock assessment can be found at Appendix B in Horn (2017).

16 Syntax conventions and examples

16.1 Input File Specification

The file format used for Casal2 is based on the command block formats used in CASAL and SPM. It is a text file that contains definitions organised into blocks.

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 configuration files:

- White space can be used freely. Tabs and spaces are both valid.
- A block ends only at the beginning of a new block or at the end of the final configuration file.
- Configuration files can include other configuration files.
- Included files are placed in-line, so a block can be continued in a new file.
- The configuration files support in-line declarations of objects.

16.2 Keywords And Reserved Characters

In order to allow efficient creation of input files, the Casal2 file format has special keywords and characters that cannot be used for labels.

Labels cannot start with a double underscore — labels with a double underscore are reserved, and are used by Casal2 for automatic reports and other internal constructs.

Block Definitions Each block in the configuration file must start with the block definition character, which is the "`@`" character.

Example:

```
@block1 <label>
type <type>

@block2 <label>
type <type>
```

The '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 Comments) Comments are supported in the configuration file on one line (to the end of that line) or over multiple lines. Comments on single lines start with the "#" character.

Example:

```
@block <label>
type <sub_type> # Descriptive comment
# parameter <value_1> *** This whole line is commented out
parameter <value_1> # <value_2> *** value_2 is commented out
```

/* */ (Multiple Line Comments) 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) Individual elements of a vector can be referenced using the { } syntax. For example, when estimating `ycs_values` a range or block of YCS values can be referenced.

Example:

```
@estimate YCS
parameter process[Recruitment].ycs_values{1975:2012}
type uniform
lower_bound
upper_bound
```

':' (Range Specifier) The range specifier ":" allows specifying a range of values instead of specifying each value explicitly. Ranges can be either incremental or decremental.

Example:

```
@process my_recruitment_process
type constant_recruitment
# With the range specifier
years_to_run 1999:2009
```

```
@process my_mortality_process
type natural_mortality
# Without the range specifier
years_to_run 2000 2001 2002 2003 2004 2005 2006 2007
```

',' (List Specifier) When a parameter supports multiple values in a single entry, the list specifier "," can be used to define multiple values as a single parameter.

Example:

```
@categories
format sex.stage
# With the list specifier
names male,female.immature,mature

@categories
format sex.stage
# Without the list specifier
names male.immature male.mature female.immature female.mature
```

'table' and 'end_table' Keyword The table keyword `table` is used to define a block of values used as parameters (e.g., catch data, observations data, etc.). In many cases an appropriate table label will need to be supplied (i.e., 'obs', 'error_value', or simply 'table', depending on where used) The first line following the `table` declaration must either (1) contain a list of columns to be used, or (2) in the case of observations the data in the specified format. The subsequent 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 its own line.

Example:

```
@block <label>
type <sub_type>
parameter <value_1>
table <table_label>
<column_label_1> <column_label_2> ...  <column_label_N>
<row1_value_1>   <row1_value_2>   ...  <row1_value_N>
<row2_value_1>   <row2_value_2>   ...  <row2_value_N>
end_table
```

[ ] (in-line Declarations) When an object takes the label of a target object as a parameter, the label can be replaced with an in-line declaration. An in-line declaration "[ ]" is a complete declaration of an object on one line. This feature is designed to allow simplifying the configuration definition.

Example:

```
@model
# With in-line declaration with label specified for time step
time_steps step_one=[type=iterative; processes=recruitment ageing]

@model
# With in-line declaration with default label (model.1)
time_steps [type=iterative; processes=recruitment ageing]
```

```
# Without in-line declaration
@model
time_steps step_one

@time_step step_one
processes recruitment ageing
```

Categories The Casal2 population representation is essentially a 2-dimensional structure. The partition is:

Categories x Ages or Lengths

Each category allows for a different range of ages or lengths and accessibility during different time periods.

Because each category can be quite complicated, the syntax for defining categories has been structured to allow for complex definitions using a simple shorthand structure.

The ”format” parameter allows for defining the structure of the category labels. Using a ”.” (period) character between each segment allows for shorthand lookups of categories.

The ”names” parameter is a list of the category names. The syntax of these names is required to match the ”format” parameter so Casal2 can organise and search on them. Using the ”list specifier” and range characters this parameter can be shortened.

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: Additional format segment not defined

names male,female.immature,mature.notag,2001:2005 # Valid
# Without the shorthand syntax these categories would be written:
names male.immature.notag male.immature.2001 male.immature.2002
male.immature.2003 male.immature.2004 male.immature.2005 male.mature.notag
  male.mature.2001 male.mature.2002 male.mature.2003 male.mature.2004
  male.mature.2005 female.immature.notag female.immature.2001
  female.immature.2002 female.immature.2003 female.immature.2004
  female.immature.2005 female.mature.notag female.mature.2001
  female.mature.2002 female.mature.2003 female.mature.2004 female.mature.2005
```

16.3 Examples of shorthand syntax and use of reserved and key characters

Categories Casal2 allows for many user-defined categories so shorthand syntax has been added to aid in the definition of complex configuration labelling and partition structures. For example, when defining categories a comma ”,” can be used to shorten lists of categories.

This syntax is the long way:

```
@categories
format sex.stage
names male.immature male.mature female.immature female.mature
```

For the exact same partition structure specified in a shorter way:

```
@categories
format sex.stage
names male,female.immature,mature
```

Casal2 requires categories in processes and observations so that the correct model dynamics can be applied to the correct elements of the partition.

An example of a process where categories are required as an input command is for ageing

```
# 1. The standard way
@ageing my_ageing
categories male.immature male.mature female.immature female.mature

# 2. The first shorthand way
@ageing my_ageing
categories male,female.immature,mature

# 3. Wild Card (all categories)
@ageing my_ageing
categories *

# 4. The second shorthand way
@ageing my_ageing
categories sex=male sex=female
```

To combine/aggregate categories together, use the ”+” special character. For example, this feature can be used to specify that the total biomass of the population is made up of both males and females.

For example,

```
@observation CPUE
type biomass
catchability Fishq
time_step one
categories male+female
selectivities FishSel
likelihood lognormal
time_step_proportion 1.0
years 1992:2001
table obs
1992    1.50    0.35
1993    1.10    0.35
1994    0.93    0.35
1995    1.33    0.35
1996    1.53    0.35
1997    0.90    0.35
1998    0.68    0.35
1999    0.75    0.35
2000    0.57    0.35
2001    1.23    0.35
end_table
```

This combination/aggregation functionality can be used to compare an observation to the total combined population:

```
@observation CPUE
type biomass
catchability Fishq
time_step one
categories *+
selectivities FishSel
likelihood lognormal
time_step_proportion 1.0
years 1992:2001
table obs
1992    1.50    0.35
1993    1.10    0.35
1994    0.93    0.35
1995    1.33    0.35
1996    1.53    0.35
1997    0.90    0.35
1998    0.68    0.35
1999    0.75    0.35
2000    0.57    0.35
2001    1.23    0.35
end_table
```

If `male` and `female` are the only categories in a population, then this is the same syntax as the command block above it.

Shorthand syntax can be useful when applying processes to a select group of categories from the partition.

For example, to apply a spawning migration to the mature categories in the partition and the partition was defined:

```
@categories
format area.maturity.tag
names north.immature.notag,2011 north.mature.notag,2011 south.immature.notag,2011
south.mature.notag,2011
```

Then, to migrate a portion of the mature population from the southern area to the northern area:

```
@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 fully or partially. An example of a parameter that is type vector is `ycs_values` in a recruitment process.

For example, a recruitment block:

```
@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 method to specify a sequence of values
# use an asterix to represent a vector of repeating integers
ycs_values 1*8
steepness 0.9
age 1
```

To estimate the last four years of the parameter `process[WestRecruitment].ycs_values` only can be specified as

```
@estimate
parameter process[WestRecruitment].ycs_values{1980:1983}
type uniform
lower_bound 0.1 0.1 0.1 0.1
upper_bound  10  10  10  10
```

Note that 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 a `@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
```

For example, to estimate only one element of the map (say 1992), and force all other years to be the same as the one estimate, can be done in the `@estimate` block using `same`:

```
@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 (see Section 4.3).

For example,

```
@observation chatCPUE
type biomass
catchability [q=6.52606e-005]
time_step one
categories male+female
selectivities chatFselMale chatFselFemale
likelihood lognormal
time_step_proportion 1.0
years 1992:2001
```

```
table obs
1992    1.50    0.35
1993    1.10    0.35
1994    0.93    0.35
1995    1.33    0.35
1996    1.53    0.35
1997    0.90    0.35
1998    0.68    0.35
1999    0.75    0.35
2000    0.57    0.35
2001    1.23    0.35
end_table

@estimate
parameter catchability[chatTANbiomass.one].q
type uniform_log
lower_bound 1e-2
upper_bound 1
```

In the above code catchability is defined and estimated without explicitly creating a `@catchability` block.

When an in-line declaration is made, the new object will be created with the name of the creator's `label.index`, where `index` is the word "`one`" through "`nine`" if it is 1 through 9, and the number if it is 10+.

For example,

```
@mortality halfm
selectivities [type=constant; c=1]

would create
@selectivity halfm.one
```

If there are 10 categories, each with its own selectivity, the 10^{th} selectivity is labelled

```
@selectivity halfm.10
```

16.4 Processes

Processes are special in how they can be defined. Typically, specifying a process is

```
@process Recruitment
type recruitment_beverton_holt
```

However, for convenience and clarity, this block can also be specified as

```
@recruitment Recruitment
type beverton_holt
```

The difference is that the keyword `process` can be replaced with the first word of the process type. In the example above this is the `recruitment` process. This option can be used to create more succinct model configurations.

More examples:

```
@mortality Fishing_and_M
type instantaneous


@transition Migration
type category
```

16.5 An example of a simple model

This example describes a 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.

The default Casal2 configuration filename is `config.csl2`. In this example, `config.csl2` specifies the files to include to run the Casal2 model from the current directory using the *!include* command.

```
## Include the input configuration files required
################################################

# Example input configuration file:
```

It is recommended to separate the sections of a Casal2 model for enhancing readability and error checking, and including the files in a version control system.

The file `population.csl2` contains the population information. The model years are from 1975 through 2012, with 3 time steps. The model is initialised over a 120 year period prior to 1975 and applies the following processes

- A Beverton-Holt recruitment process, recruiting a constant number of individuals to the first age class (i.e., $age = 1$).
- A constant mortality process representing natural mortality(M). This process is repeated in all 3 time steps, so that a proportion of M is applied in each time step.
- 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 3 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 attributes for 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 third time step applies .

The first 28 lines of the main section of the `population.csl2`:

```
########################################
# The Population definition for the model
########################################

# The model definition. (This must be the first @command in the config files)
@model
start_year 1975
final_year 2012
projection_final_year 2025
min_age 1
```

```
max_age 30
age_plus true
base_weight_units tonnes
initialisation_phases Equilibrium_state
time_steps Sep_Feb Mar_May Jun_Aug

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

@initialisation_phase Equilibrium_state
type derived

# Define the processes in the Annual Cycle
@time_step Sep_Feb
processes Recruitment Instantaneous_Mortality
```

To run the model to verify that the model runs without any syntax errors, use the command `casal2 -r`. Since Casal2 reads in the default filename `config.csl2`, this filename can be overridden. For example, if the model is in file `Mymodel.txt`, then this filename would be specified using the `-c` option, `casal2 -r -c Mymodel.txt`.

To 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`. The output has been redirected to file `estimate.log` using the command `casal2 -e > estimate.log`. Reports for the user-defined reports `reports.csl2` from the final iteration of the estimation are output to the file `estimate.log`, and successful convergence is printed to the screen

```
Total elapsed time: 1 second
Completed
```

The main output from the estimation run is summarised in the file `estimate.log`, and the final MPD parameter values can also be redirected 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 be run, using `casal2 -p > profile.log`. See the examples folder for the example of the output.

17 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.

Casal2 has two **R** packages, a base library which is bundled with Casal2 application and a post processing package r4Casal2 for plotting and model comparisons https://github.com/NIWAFisheriesModelling/Casal2_contrib. The base **R** package is made to read and write output from Casal2 where as the post-processing package is more generalisable.

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
- extract.csl2.file(), which reads a Casal2 .csl2 (configuration) file into a list
- write.csl2.file(), which writes a Casal2 .csl2 (configuration) file to a file

- `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.

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")
```

18 Troubleshooting

Casal2 can implement complex models that provide many opportunities for error — either because the parameter files do not correctly specify the model, or because the model specified does not appear to work as expected. When in doubt, ask an experienced user. Debugging versions of Casal2 are available that can help to track down cryptic errors.

If you cannot resolve an issue using these guidelines then please contact the development team. To report an issue please follow the guidelines described in Section 18.2.1.

For most issues, Casal2 attempts to produce informative error messages. There are optional command line arguments that will give more verbose reporting, and should enable additional information to help resolve a problem. However, when Casal2 generates an error and the error message does not make sense, please let the Development Team know. Even if you manage to fix the problem yourself, we may be able to implement a more helpful error message or modify the user manual, and make life easier for the next person to encounter the problem. You can do this by submitting an issue in the GitHub repository at `https://github.com/NIWAFisheriesModelling/CASAL2/issues`.

18.1 Logging

Casal2's internal logging system can be invoked at the command line with argument `--loglevel` followed by one of the options: `trace`, `finest`, `fine`, `medium`.

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. The ordering for the options is that `medium` is 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.

For example, to enable logging with trace level output:

- On Windows: `casal2 -r --loglevel trace > run.log 2> run.err`
- On Linux: `casal2 -r --loglevel trace > run.log 2&> run.err`

This argument will output Casal2's reports to the file "run.log", and the "`2>`" or "`2&>`" syntax will print the error logged information to the file "run.err". You should be able to see where Casal2 failed, and is exited by going to the end of the "run.err" file and looking at the last few messages.

18.2 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 behaviour as well as the expected behaviour. If possible, try to reduce the input configuration file to demonstrate the error with a reduced set of commands and model structure, and aim to have as little else going on in the model as possible. This will make it faster to isolate the problem and provide a solution or fix.

18.2.1 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 v24.10 (2024-10-07)". The version is output by Casal2 with the command `casal2 -v`.
3. Provide the system you are using, e.g., "x64 Intel CPU with Microsoft Windows 10".
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` 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.

19 Casal2 software license

GNU GENERAL PUBLIC LICENSE

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

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.

20 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, the methods and equations, and documentation draw heavily on similar components of the fisheries population modelling application CASAL (Bull et al., 2012) and the spatial population model SPM (Dunn et al., 2021). We thank the authors of CASAL and SPM for their permission to use their work as the basis for parts of Casal2 and allow the use of the definitions, concepts, and documentation.

The development of Casal2 was funded by the New Zealand Ministry for Primary Industries and the National Institute of Water & Atmospheric Research Ltd. (NIWA). More recent developments of this version were funded by Ocean Environmental Ltd.

21 References

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.

I.J. Doonan, K. Large, A. Dunn, S. Rasmussen, C. Marsh, and S. 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 v2.03-2021-06-03. Technical report, Ocean Environmental, 2021.

A. Dunn, A. Grüss, J.A. Devine, C. Miller, P. Ziegler, D. Maschette, T. Earl, C. Darby, and F. Massiot-Granier. Integrated toothfish stock assessments using casal2. Technical Report WG-SAM-2022/14, CCAMLR, 2022.

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

R.I.C.C. Francis, V. Haist, and B. Bull. Assessment of hoki (*Macruronus novaezelandiae*) in 2002 using a new model. *New Zealand Fisheries Assessment Report*, 6, 2003.

A.B. Gelman, J.S. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian data analysis*. Chapman and Hall, London, 1995.

W.R. Gilks, A. Thomas, and D.J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

P.L. 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.

Kai Lorenzen, Edward V. Camp, and Taryn M. Garlock. Natural mortality and body size in fish populations. *Fisheries Research*, 252:106327, 2022. ISSN 01657836. doi: 10.1016/j.fishres.2022.106327. URL https://linkinghub.elsevier.com/retrieve/pii/S0165783622001047.

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

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation: Special Issue on Uniform Random Number Generation*, 8(1):3–30, January 1998.

M.N. Maunder. A review of integrated analysis in fisheries stock assessment. *Fisheries Research*, page 14, 2013.

A.E. Punt and R.W. 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/.

C. Sherlock and G. Roberts. Optimal scaling of the random walk metropolis on elliptically symmetric unimodal targets. *Bernoulli*, 15(3), 2009.

R. Storn and K. 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.

J.T. Thorson, K.F. Johnson, R.D. Methot, and I.G. Taylor. Model-based estimates of effective sample size in stock assessment models using the dirichlet-multinomial distribution. *Fisheries Research*, 192:84–93, 2017.

A. Wächter, A. Kowarz, and A. Griewank. Adol-c: a package for the automatic differentiation of algorithms written in c/c++. *ACM TOMS*, 22(2):131–167, 1996.

C.J. Walters and D. Ludwig. Calculation of bayes posterior probability distributions for key population parameters. *Canadian Journal of Fisheries and Aquatic Sciences*, 1994.

22 Index

Appendices

A Compiling Casal2

This section describes how to set up the environment on your local machine that will allow you to build and compile Casal2. The build environment can be on either Microsoft Windows or Linux systems. At present the Casal2 build system supports Microsoft Windows 10+ and Linux (with GCC/G++ 4.9.0+). Apple OSX or other platforms are not currently supported.

A.1 Overview

The build system is made up of a collection of Python scripts that do the various tasks. These are located in `CASAL2/BuildSystem/buildtools/classes/`. Each Python script has its own set of functionality and undertakes one set of actions.

The top level of the build system can be found at `CASAL2/BuildSystem/`. In this directory you can run `doBuild.bat help` from a command terminal in Microsoft Windows systems or `./doBuild.sh help` from a terminal in Linux systems.

The script will take one or two parameters depending on what style of build you want to undertake. These commands allow the building of various stand-alone binaries, shared libraries, and the documentation. Note that you will need additional software installed on your system in order to build Casal2. The software requirements are described below.

A summary of all of the doBuild arguments can be found using the command `doBuild help` in the BuildSystem directory.

The arguments to doBuild are:

Usage: doBuild <build_target> <argument>

`help` Print out the doBuild help (this output)

`check` Do a check of the build system

`clean` Remove debug/release build files

`clean_all` Remove all build files and ALL prebuilt binaries

`version` Build the current version files for C++, R, and LaTeX

Build required libraries (DLLs/shared objects for Casal2)

`thirdparty` Build the third party libraries

<option> Optionally specify the target third party library to build, either adolc or betadiff (default is none)

Build development and test versions (for development builds only)

`release` Build stand-alone release executable

<option> Optionally specify the target third party library to build, either adolc or betadiff (default is none)

`debug` Build stand-alone debug executable

<option> Optionally specify the target third party library to build, either adolc or betadiff (default is none)

`test` Build stand-alone unit tests executable

`unittests` Run the unit tests (requires that 'test' has been built)

`modelrunner` Run the test case models

Build the Casal2 end-user application

`library` Build shared library for use by front end application

<argument> Required argument to specify the target library to build: release, adolc, betadiff, or test

`frontend` Build Casal2 front end application

Create the archive, R Library, documentation, and the installers

`documentation` Build the Casal2 user manuals

`rlibrary` Create the R library

`archive` Create a zipped archive of the Casal2 application.

<true> if specified build skips everything but frontend

`installer` Create the Microsoft Windows installer package

`deb` Create Linux Debian installer

The outputs from the build system commands will be placed in sub-folders of `CASAL2/BuildSystem/bin/<operatingsystem>/<build_type>`

For example:

`CASAL2/BuildSystem/windows/debug`

`CASAL2/BuildSystem/windows/library_release`

`CASAL2/BuildSystem/windows/thirdparty/`

`CASAL2/BuildSystem/linux/library_release`

The files `Casal2_build.bat` for Windows and `Casal2_build.sh` for Linux in the root folder contain all the calls in the correct order of `doBuild` required to successfully build Casal2, the documentation, the Windows installer (Windows) or the Debian installer (Linux), the R-Libraries, and run all the test cases and unit tests.

A.2 Building on Microsoft Windows

A.2.1 Prerequisite software

The building of Casal2 requires additional build tools and software, including Python, git version control, GCC compiler, LaTeX compiler, and a Windows package builder. Casal2 can require specific implementations of these packages and versions in order to build without modifying the build scripts.

C++ and Fortran compiler Source: tdm-gcc (MinGW-w64) from `https://jmeubank.github.io/tdm-gcc/`.

Casal2 is designed to compile under GCC on Microsoft Windows and Linux. While it may be possible to build the package using different compilers, the Casal2 Development Team does provide any assistance or recommendations. We recommend using 64-bit TDM-GCC with a version of at least 10.3.0. Ensure you have the "fortran" and "openmp" options installed as a part of the "gcc" install option drop-down tick boxes as these are required. For example, from `https://jmeubank.github.io/tdm-gcc/articles/2021-05/10.3.0-release`, select the 64+32-bit MinGW-w64 edition, then select the Custom install and tick all boxes.

Note that a common error that can be made is having a different GCC compiler in your path when attempting to compile. For example, `rtools` includes a version of the GCC compiler. We recommend removing these from your path prior to compiling.

GIT version control Source: Command line GIT from `https://www.git-scm.com/downloads`.

Casal2 automatically adds a version details to its files and any output based on the GIT version of the latest commit to its repository. This includes the name of source repository that was used. The command line version of GIT is used to generate the version details.

MiKTeX LaTeX Processor Source: Portable version from `http://www.miktex.org/portable`.

The main user documentation for Casal2 is a PDF document generated from LaTeX. The LaTeX syntax sections of the documentation are generated, in part, directly from the code. In order to generate the user documentation, you will need the MiKTeX LaTeX compiler.

A number of additional LaTeX styles are used — these will usually be identified doing the doBuild process and can be installed as required.

7-Zip Source: 7-Zip from `http://www.7-zip.org/download.html`.

The build system calls `7zip.exe` to unzip files in the build system; it is advised to have this in the path.

Python Source: Python3 from `https://www.python.org/downloads/windows/`

Python is used to run the build scripts and set the required environment variables required to build Casal2.

Python modules There are a number of Python3 modules that are required to build Casal2. These can be installed with `python -m pip install` *`module-name`*. For example, You may need to install `datetime`, `re`, and `distutils` Python modules.

Inno setup installer build (optional) Source: Inno Setup 5 from `http://www.jrsoftware.org/isdl.php`

If you wish to build a Microsoft Windows compatible Installer for Casal2 (recommended) then you will need the 'Inno Setup 5' application installed on the machine. The installation path must be `C:\ProgramFiles(x86)\InnoSetup5\` in order for the build scripts to find and use it.

A.2.2 Pre-build requirements

Prior to building Casal2 you will need to ensure you have both G++ and GIT in your path. You can check both of these by typing the following commands and checking that they return the correct version number:

```
g++ --version
```

```
git --version
```

This also allows you to check that there are no alternative versions of a GCC compiler that may confuse the Casal2 build. It's also worth checking to ensure GFortran has been installed with the G++ compiler by typing:

```
gfortran --version
```

If you wish to build the documentation, `bibtex` will also need to be in the path, e.g., to check, try:

```
bibtex --version
```

A.2.3 Building Casal2

The build process is relatively straightforward. Before you start the build process, you can run `doBuild check` from the command prompt to check if your build environment is complete. Make sure that you are within `CASAL2/BuildSystem/` to run `doBuild`.

`doBuild check` will summarise Windows environment PATH as a part of its output, and this can be used to check that the paths for g++ and gfortran and the g++ point to where the correct version of GCC is installed.

The build process is as follows:

1. Download a clone of the code on your local machine
2. Navigate to the BuildSystem folder in `CASAL2/BuildSystem`
3. You need to build the third party libraries with the following commands from the command prompt:
   - `doBuild thirdparty`
4. You need to build the binary you want to use:
   - `doBuild release`
5. You can build the documentation if you want:
   - `doBuild documentation`

A.3 Building on Linux

This guide has been written against a fresh install of Ubuntu 20.04. With Ubuntu we use apt-get to install new packages. You'll need to be familiar with the package manager for your distribution to correctly install the required prerequisite software. For this you will require administrator level access.

A.3.1 Prerequisite software

G++ compiler If gfortran is not installed, install this with: `sudo apt-get install gfortran`.

GIT version control Git may not be installed by default and it can be installed with `sudo apt-get install git`

Casal2 automatically adds a version details to its files and any output based on the GIT version of the latest commit to its repository. This includes the name of source repository that was used. The command line version of GIT is used to generate the version details.

CMake CMake is required to build multiple third-party libraries and the main code base. You can do this with `sudo apt-get install cmake`

Python Python3 is used to run the build scripts and set the required environment variables required to build Casal2. This is usually installed by default on Linux systems, but if not, it can be installed using: `sudo apt-get install python3`

Python modules There are a number of Python3 modules that are required to build Casal2. These can be installed with `sudo apt-get install module-name`. For example, You may need to install `datetime`, `re`, and `distutils` Python modules.

LaTeX LaTeX on Linux is required, and the Texlive LaTeX Processor is recommended. This can be installed with:

```
sudo apt-get install texlive-binaries  sudo apt-get install texlive-latex-base
sudo apt-get install texlive-latex-recommended           sudo apt-get install
texlive-latex-extra
```

Alternatively you can install the complete package with `sudo apt-get install texlive-full`

A number of additional LaTeX styles are used — these will usually be identified doing the doBuild process and can be installed as required.

A.3.2 Building Casal2

The build process is relatively straightforward. You can run `./doBuild.sh check` to see if your build environment is ready.

1. Download a clone of the code on your local machine
2. Navigate to the BuildSystem folder in `CASAL2/BuildSystem`
3. You need to build the third party libraries with:
    - `./doBuild.sh thirdparty`
4. You need to build the binary you want to use:
    - `./doBuild.sh release`
5. You can build the documentation:
    - `./doBuild.sh documentation`

A.4 Troubleshooting

A.4.1 Third-party C++ libraries

It's possible that there will be build errors or issues building the C++ third-party libraries. If you encounter an error, then check the log files to locate the source of the problem. Each third-party build system stores a log of everything that was done. The files will be named

- casal2_unzip.log
- casal2_configure.log
- casal2_make.log
- casal2_build.log
- . . . etc,.

Some of the third-party libraries require very specialised environments for compiling under GCC on Windows. These libraries are packaged with MSYS (MinGW Linux style shell system). The log files for these will be found in `ThirdParty/<libraryname>/msys/1.0/<libraryname>/`

e.g., `ThirdParty/adolc/msys/1.0/adolc/ADOL-C-2.5.2/casal2_make.log`
e.g., `ThirdParty/boost/boost_1_58_0/casal2_build.log`

A common issue when running doBuild thirdparty are Python error messages about missing modules, e.g., ModuleNotFoundError: No module named 'dateutil'. This type of error message indicates that a Python module (library) is missing and will need to be installed. For instance, to

install the 'dateutil' module, type the following into a command prompt or terminal window: pip3 install python-dateutil.

A.4.2 Main code base

If the unmodified code base does not compile, the most likely cause is an issue with the third-party libraries not being built correctly. As updates and revisions are outside the control of the Development Team, problems can arise that may require the developers of the third party libraries to resolve first. However, versions of these libraries are included in the Casal2 source code and these should work. For any specific issues contact a local expert with regard to your specific system environment, or else the Casal2 Development Team for help.

B Casal2 build guidelines and validation

B.1 Casal2 coding practice and style

Casal2 is written in C++ and uses the Google C++ style guide (see `https://google.github.io/styleguide/cppguide.html`).

In general when editing or writing code for Casal2

1. Using consistent indentations inside functions and loops, and descriptive and human readable variable or function names.
2. Use of the characters '_' on the end of class variables defined in the .h files.
3. Annotate and comment the code, especially where it would help another contributor understand the program logic and rationale.
4. Add descriptive log messages to aid in debugging and checking of the program logic flow.
5. Implement unit tests, internal models, and external models to test and validate the new or changed functionality.
6. Document the functionality in the Casal2 User Manual(s).

Casal2 allows printing of logging messages art runtime using the –loglevel command line argument. The levels of logging in Casal2 are:

- LOG_MEDIUM() usually reserved for iterative functionality (e.g. estimates during estimation phase)
- LOG_FINE() the level of reporting between an actual report and a fine scale detail that end users are not interested in (Developers)
- LOG_FINEST() Minor fine scale details within a function or routine.
- LOG_TRACE() put at the beginning of every function

e.g., to run Casal2 with logging, use

```
casal2 -r --loglevel finest > my_run.log 2> my_run.err
```

This will output all the logged information to `my_run.err`.

B.2 Units tests and model validation

The Casal2 development places an emphasis on maintaining software integrity and reproducibility between revisions. Casal2 uses model validations and built in unit tests to validate and verify the code each time Casal2 is compiled and built.

There are three different validation approaches in Casal2. These are:

1. Mocking specific classes.
2. Implementing internal models (implemented in C++ source code with extension `.Test.cpp`) that have variable test cases for specific classes.
3. Implementing externally run models (found in the TestModel folder) that are validated to generate expected output.

To implement mocking of classes and internal models, Casal2 uses the Google testing framework and the Google mocking framework.

To implement testing of full models, input configuration files are run using the compiled Casal2 binaries, and the output compared with expected output using @assert commands.

B.2.1 Mocking specific classes

Classes are unit tested using unit tests that are a part of the source code. These are designed to check the components of the code to validate that functions provide expected output. These unit tests are run each time Casal2 is compiled.

When adding unit tests, they should to be developed and tested outside of Casal2ṪThis gives confidence that the test does not contain any calculation errors.

Mocking specific classes is used to validate specific functionality and is encouraged because it is the easiest to isolate simple errors that may be introduced with code changes.

As examples, see (i) the file `VonBertalanffy.Test.cpp` mocks the von Bertalanffy age-length class and tests the mean length calculation, and (ii) the `Partition` class has the file `Partition.Test.cpp` that validates user inputs and model expectations.

B.2.2 Internal mocking of simple models

Mocking of simple models is done using a number of internal models. Most of the functionality for implementing these are in the source folder `/CASAL2/source/TestResources`.

These implements simple models and run test cases with differing class implementations by running an internal empty model and testing the output of classes from a model run.

As examples, see (i) the `LogNormal.Test.cpp` in the `Projects` class that test the lognormal distribution when used for projections, and (ii) the `TagByLength` process in `TagByLength.Test.cpp` that tests functionality of the tagging process.

B.2.3 External testing using test models

External tests are run following compilation using the Python modelrunner.py scripts (i.e., using the `DoBuild modelrunner` script in the BuildSystem folder). These models are used to test model runs, minimisation routines, and MCMC output.

The test model input configuration files are located in the `TestModel` folder and the command calls to run these are in the modelrunner.py script.

Contributors are encouraged to add additional models to the list of test models as these be used to validate the combined functionality of a range of interrelated commands and subcommands in Casal2.

B.3 Verification

After Casal2 executes Validate and Build it runs sanity checks in the verify state. These are business rules that can be checked across the entire system. This can be useful to suggest dependencies or configurations. For an example see the directory `Processes\Verification\` in the source code.

B.4 Reporting (optional)

Currently Casal2 has reports that are **R** compatible, i.e., all output reports produced by Casal2 can be read into **R** using the standard **CASAL2 R** package. If you create a new report or modify an old one, you most follow the standard so that the report is **R** compatible.

All reports must start with, `*label (type)` and end with, `*end`

Depending on what type of information you wish to report, will depend on the syntax you need to use. For example

{d} (Dataframe) Report a dataframe

```
*estimates (estimate_value)
values {d}
process[Recruitment_BOP].r0 process[Recruitment_ENLD].r0
2e+006 8e+006
*end
```

{m} (Matrix) Report a matrix

```
*covar (covariance_matrix)
Covariance_Matrix {m}
2.29729e+010 -742.276 -70160.5
-110126 -424507 -81300
-36283.4 955920 -52736.2
*end
```

{L} (List) Report a List

```
*weight_one (partition_mean_weight)
year: 1900
ENLD.EN.notag {L}
mean_weights {L}
0.0476604 0.111575 0.199705
end {L}
age_lengths {L}
12.0314 16.2808 20.0135
end {L}
end {L}
*end
```

B.5 Update the Casal2 User Manual(s)

Contributors will need to add or modify sections of the user manual(s) to document their changes. This includes the section that describes the methods and the section where the specific syntax is defined.

B.6 Builds to pass before merging changes

Once you have made changes, you must run the following before your changes can be included in the master code.

- Build the unittest version. See Section A for how to build unittest depending on your system.
- Run the standard and new unit tests to check that they all pass. To do this first compile the test executable using the script `DoBuild test`. Then move to the directory with the location of the executable (`BuildSystem/bin/OS/test`) and run it (open a command terminal and run `casal2`) to check all the unit-tests pass.
- Test that the debug and release of Casal2 compiles and runs with `DoBuild debug`
- Run the second phase of unit tests (requires that the debug version is built). This runs the tests that comprise of complete model runs using `DoBuild modelrunner`

- Build the archive using `DoBuild archive` which builds all required libraries. There are small nuances between Double classes, especially when reporting the class that mean seemingly simple changes can sometimes cause a break in the full build.