C++ Agent Based Model (CABM) User Manual

C. Marsh

2022

CABM User Manual (modified 2022-05-16) for use with
`cabm-v2022-05-16 (rev. 503df7db)`

Citation: C. Marsh (2022). C++ Agent Based Model (CABM) User Manual, v2022-05-16 (rev. 503df7db). . 152 p.

Contents

1. Introduction

CABM is a command line program that implements a generalised agent-based model (ABM). An ABM is a model that represents a fish stock as a collection of agents. An agent is defined as one or more fish with homogenous characteristics, i.e. length, weight and sex. When an agent represents a single individual, the ABM becomes an individual-based model (IBM) (Grimm and Railsback, 2013). Given fish stocks consist of millions if not billions of individuals, ABMs are often more practical than IBMs due to computational limitations, i.e. it requires large amounts of memory to record and modify millions of agents. ABMs use functions to grow, move, create and kill agents over time, termed agent dynamics. When summaries are made over all agents, stock level quantities are observed. Simulating stocks with this high level of detail allows heterogeneity in key dynamics such as growth and mortality. This is an advantage of ABMs as this heterogeneity is often approximated in other operating models (OM).

CABM was created to emulate a stock over a fine spatial resolution domain, apply realistic movement and replicate complex fishing processes. These were all attributes of an OM that are of interest for exploring stock assessment methods.

CABM was designed to be flexible with respect to specifying agent dynamics, spatial resolution, timing of agent dynamics, simulated observations and model outputs. It was designed for emulating fish populations, but can be modified to represent other populations. CABM is a program I developed during my PhD and I hope that it will be useful to others. The core modelled entity is an agent. An agent can represent many individual fish with identical characteristics.

CABM may not be as efficient when compared to bespoke agent-based models in the literature. This is because generality and flexibility comes with a computational cost, but I have found the program to be fast enough for most tasks that I have used it for. The bonus in generality is it has readable input syntax and error handling making it faster to develop and run models (in Theory).

CABM applies an annual cycle each model year for a user defined number of years. The annual cycle consists of user defined number of discrete time-steps. In each time-step users must specify the agent dynamics, simulated observations and model outputs. It can simulate many different user defined quantities, for example removals-at-length or -age from an anthropogenic or exploitation event (e.g. fishery or other human impact), scientific survey and other biomass indices, and mark-recapture data.

The real power of CABM is its use in a closed loop management strategy evaluation (MSE). Recent code changes have allowed CABM to run a model run and then once it reaches assessments years it will run an R script and wait for future catches before continuing to the next assessment year (see Section 8 for more details).

1.1. Where to get CABM

CABM source code is hosted on GitHub, and can be found at `https://github.com/Craig44/CABM`.

Currently you can either compile the code using the `BuildSystem`, or download a pre-compiled version from `https://github.com/Craig44/CABM/releases`. This repository is self contained with all the required third-party libraries and has been developed for ease of compilation (it is very easy see the GitHub page for information).

1.2. System requirements

CABM is available for most IBM compatible machines running 64-bit Linux and Microsoft Windows operating systems, unfortunately Mac has not been tested.

Several of CABM's tasks are highly computer intensive and a fast processor is recommended. Depending on the model implemented, some of the CABM tasks can take a considerable amount of processing time.

The program itself can require anywhere from a few gigs to 10's of gigs of hard-disk space, depending on the size and complexity of the model. Output files can also consume large amounts of disk space. Depending on the number and type of user output requests, the output could range from a few hundred kilobytes to several hundred megabytes. Several hundred megabytes of RAM may be required, depending on the spatial size of the model, number of agents, and complexity of processes and observations. For extremely large models, several gigabytes of RAM may occasionally be required.

1.3. Necessary files

For both 64-bit Linux and Microsoft Windows, only the binary executable `cabm` or `cabm.exe` is required to run CABM.

CABM comes with an **R** (R Core Team, 2014) package to assist in the post processing of CABM output. We provide the `ibm` **R** package for importing the CABM output into **R** (see Section 14).

1.4. Getting help

CABM is distributed as 'officially' unsupported software although I am always happy to help any user who wants to give this a crack. The Development Team would appreciate being notified of any problems or errors in CABM, please use the github page to post issues, see Section 16.2 for the recommended template for reporting issues.

1.5. Technical details

CABM was compiled on Linux using `gcc` (`http://gcc.gnu.org`), the C/C++ compiler developed by the GNU Project (`http://gcc.gnu.org`). **note** this program uses OpenMP which is an option that you should tick if you want to compile the code. The 64-bit Linux version was compiled using `gcc` version 5.1.0 20151010 Ubuntu Linux (`http://www.ubuntu.com/`). The Microsoft Windows (`http://www.microsoft.com`) version was compiled using MingW (`http://www.mingw.org`) `gcc` (tdm64-1) 5.1.0 (`http://gcc.gnu.org`). The Microsoft Windows(`http://www.microsoft.com`) installer was built using the Inno Setup 5 (`http://www.jrsoftware.org/isdl.php`).

The random number generator used by CABM uses an implementation of the Mersenne twister random number generator (Matsumoto and Nishimura, 1998). This, the command line functionality, matrix operations, and a number of other functions use the BOOST C++ library (Version 1.58.0). The threading capabilities are done using [OpenMP library]

1.6. The future for CABM

- Make it faster...

- ~~Add multiple fisheries at the same time, often we have multiple fleets fishing e.g trawler and long-liners, currently we can have multiple fishing events but they do not occur simultaneously. This would have issues on data generation because the order of operations would be important and would get a bit fidgety.~~

- ~~test on a high performance computer (HPC) to see the capabilities if a user has access to this. how spatial can we go if we have threads of cores available, it would be cool to have 1000x1000 model. or alternatively use the threads in another space such as run multiple species in parallel and synchronise threads when they interact.~~. I have explored multi-threading, but found unless the model has 1000's of cells the overhead and complexity in code has not been worth continuing.

- see it used in practice other than by my self, I am hoping that this could be used as a management strategy evaluation (MSE) tool. This has the ability to test management actions on a large scale such as test structural uncertainty in our models for MSE. Include the possible effects of climate change in the future etc. Watch this space.

2. Model overview

2.1. Introduction

CABM is run from the console window in Microsoft Windows or from a terminal window in Linux. CABM gets its information from input text configuration files, the default file CABM will look for is *config.ibm*, although you can override this using the `-c` command line parameter (See Section 3.4). Commands and subcommands in the input configuration file are used to define the model structure, provide observations, define parameters, and define the outputs (reports) for CABM. Command line switches tell CABM the run mode and where to direct its output. See Section 3 for details.

We define the model in terms of the *state*. The state consists of a few key components the agents that collectively make up the *partition*, and any *derived quantities* and the spatial resolution. The state will typically change in each *time-step* of every year, depending on the *processes* defined for those time-steps in the model.

CABM expresses a fish stock as a collection of spatially referenced agents within a discrete spatial domain with R cells. Agents in spatial cell indexed by r ($r = (1,\ldots,R)$) are denoted by $\boldsymbol{X_r} = \{X_{r,1},\ldots,X_{r,n_r}\}$, where n_r is the number of agents in cell r. Agent $X_{r,i}$ is also a set where each element in $X_{r,i}$ is an agent attribute, as outlined in Table 4.1.

CABM applies an annual cycle each year which is then repeated over a specified number of years. The annual cycle is made up of discrete time blocks hereinafter referred to as time-steps. Each time-step applies a sequence of agent dynamics (Section **??**), outputs summaries of agent information and simulates synthetic data. Agent dynamics use agent specific attributes (Table 4.1) along with other assumed parameter values to define interactions and outcomes for agents over time. CABM assumes a closed spatial domain, meaning, no agents can leave the spatial extent (i.e. no immigration or emigration).

2.2. The population section

This section (Section 4) discusses how to set up the process model, which controls how agents move, die and get created through out the model time frame. It also give details on what each process does and how the syntax looks in a configuration file.

2.3. The observation section

This section (Section 6) talks about what observations CABM can create and then simulate from. This provides examples on simulated observations that can be produced by CABM. information and syntax examples.

2.4. The report section

This section (Section 7) discusses how to print output from the model, no reports are printed by default so it is important that you look at this section.

3. Running CABM

CABM is run from the console window (i.e., the command line) on Microsoft Windows or from a terminal window on Linux. CABM uses information from input data files – the *input configuration file* being the key file.

The input configuration file is compulsory and defines the model structure, processes, observations, parameters, and the reports (outputs) requested. The following sections describe how to construct the CABM configuration file. By convention, the name of the input configuration file ends with the suffix `.ibm`. However, any file name is acceptable. Note that the input configuration file can 'include' other files as a part of its syntax. Collectively, these are called the input configuration file.

Other input files can, in some circumstances, be supplied depending on what is required. For example adding additional layers so you do not clutter a single file and improve readability.

3.1. Using CABM

To use CABM, open a console (i.e. the command prompt) window (Microsoft Windows) or a terminal window (Linux). Navigate to a directory of your choice, where your input configuration files are located. Then enter `cabm` with any arguments (see Section 3.4 for the the list of possible arguments) to start your CABM job running. CABM will print output to the screen and return you to the command prompt when the job has completed. Note that the CABM executable (binary) and shared libraries (extension `.dll`) must be either in the same directory as the input configuration files or in your systems `PATH`. The CABM installer (**doesn't exist yet**) should update your path on Windows in any case, but see your operating system documentation for help identifying or modifying your `PATH`.

3.2. The input configuration file

It is advised to view input configuration files, using Text Pad 8 (`https://www.textpad.com/download/`). We provide a syntax highlighter for this program which can be found in the Documentation/UserManual called `cabm.syn`, this makes viewing input configuration files a lot easier. Or `ABM.xml` found in the Documentation directory.

The input configuration file is made up of three broad sections; the description of the population structure and parameters (the population section), the observations and their associated likelihoods (the observation section), and the outputs and reports that CABM will return (the report section). The input configuration file is made up of a number of commands (many with subcommands) which specify various options for each of these components.

The command and subcommand definitions in the input configuration file can be extensive (especially when you have a model that has many observations), and can result in a input configuration file that is long and difficult to navigate. To aid readability and flexibility, we can use the input configuration file command !*include file* (e.g. Figure 3.1). The command causes an external file, *file*, to be read and processed, exactly as if its contents had been inserted in the main input configuration file at that point. The file name must be a complete file name with extension, but can use either a relative or absolute path as part of its name. Note that included files can also contain !*include* commands. See Section 12 for more detail. I often use the file extension name `.ibm` to help me identify files associated with this program, and also to associate syntax highlighters to ease

readability. You do not have to do this, you can name your files `.txt`, `.afile` or whatever

```
1  !include "population.ibm"
2  !inlcude observation.ibm
3  !include "../Estimation.ibm"
```

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

3.3. Redirecting standard output

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

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

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

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

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

```
cabm [arguments] > out
```

e.g.

```
cabm -r > out
```

And, on some Microsoft Windows systems (e.g., Windows10), you can redirect to both standard output and standard error, using the syntax,

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

Note that CABM outputs a few lines of header information to the output (e.g. Figure **??**). The header consists of the program name and version, the arguments passed to CABM from the command line, the date and time that the program was called (derived from the system time), the user name, and the machine name (including the operating system and the process identification number). These can be used to track outputs as well as identifying the version of CABM used to run the model.

3.4. Command line arguments

The call to CABM is of the following form:

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

where,

-c *config_file* Define the input configuration file for CABM (if omitted, CABM looks for a file named `config.ibm`)

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)

-s [--simulation] number for a single set of parameters generate n number of simulated observations.

-i [--input] file_name an input file-name where users can update/overwrite parameters in the system

-v [--version] Display the CABM version number

-m [--mse] Run Management strategy evaluation mode

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

and where the following optional arguments [*options*] may be specified,

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

--loglevel arg = {trace, finest, fine, medium} (see Section 7)

3.5. Constructing the CABM input configuration files

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

Population input (Section 4) specifies the model structure, population dynamics, and other associated parameters;

Observation input (Section 6) contains all the observations data available to the model and describes how the observed values should be formatted, how CABM calculates the expected values, and the likelihoods available for each type of observation; and

Report input (Section 7) specifies any output required.

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

3.5.1. Commands

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

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

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

```
@process NaturalMortality
or
!include MyModelSpecification.csl2
```

3.5.2. Subcommands

CABM subcommands are used for defining options and parameter values related to a particular command. Subcommands always take an argument which is one of a specific *type*. The argument *types* acceptable for each subcommand are defined in Section 12, and are summarised below.

Like commands (`@command`), subcommands and their arguments are not order specific — except that that all subcommands of a given command must appear before the next `@command` block. CABM may report an error if they are not supplied in this way. However, in some circumstances a different order may result in a valid, but unintended set of actions, leading to possible errors in your expected results.

The argument type for a subcommand can be either:

switch	true/false
integer	an integer number,
integer vector	a vector of integer numbers,
integer range	a range of integer numbers separated by a colon, e.g. 1994:1996 is expanded to an integer vector of values (1994 1995 1996),
constant	a real number (i.e. double),
real vector	a vector of real numbers (i.e. vector of doubles),
real	a real number that can be estimated (i.e. float),
addressable	a real number that can be referenced but not estimated (i.e. addressable double),
addressable vector	a vector of real numbers that can be referenced but not estimated (i.e. vector of addressable doubles),
string	a categorical (string) value, or
string vector	a vector of categorical values.

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

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

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

Note that parameters defined as addressable with the subcommand type `addressable` or `addressable vector` are usually derived IBM and are not directly estimable. As such, they can be acted upon by the model (e.g. called by various processes; have priors and/or penalties assigned to them), but they do not directly contribute to any estimation within the model

3.5.3. The command-block format

Each command-block consists of a single command (starting with the symbol `@`) and, for most commands, a unique label or an argument. Each command is then followed by its subcommands and their arguments, e.g.,

`@command`, or	`@command argument`, or	`@command label`
`subcommand argument`	`subcommand argument`	`subcommand argument`
`subcommand argument`	`subcommand argument`	`subcommand argument`
.	.	.
.	.	.
etc.	etc.	etc.

Blank lines are ignored, as is extra white space (i.e., 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.

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

Commands, sub-commands and arguments in the input configuration files are not case sensitive. Labels and variable values are case sensitive. Also, on a Linux system, external calls to files are case sensitive (i.e., when using !*include file*, the argument *file* will be case sensitive).

3.5.4. Commenting out lines

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

Alternatively, to comment out an entire block or section place a /* as the first character on the line to start the comment block, then end it with */. All lines (including line breaks) between /* and */ inclusive are ignored.

```
# This is a comment and will be ignored
@process NaturalMortality
m 0.2
```

```
/*
This block of code
is a comment and
will be ignored
*/
```

3.5.5. Determining CABM parameter names

When CABM processes an input configuration file it translates each command block and each subcommand block into a unique CABM object, each with a unique parameter name. For commands, this parameter name is simply the command label. For subcommands, the parameter name format is either:

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

`command.subcommand` if the command has no label, or

`command[label].subcommand{i}` if the command has a label and the subcommand arguments are a vector, and we are accessing the *i*th element of that vector.

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

The unique parameter name is used to reference that unique parameter when, for example, estimating, applying a penalty, projecting, time varying or applying a profile. For example, the parameter name of the Natural mortality rates subcommand `m` of the command `@process` with the label `NaturalMortality` is category related and so, the syntax to reference all m related categories is,

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

Or, the syntax to specify a single category for which to apply the natural mortality process is,

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

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

3.6. CABM exit status values

Whether CABM completes its task successfully or errors out gracefully, it returns a single exit status value 'completed' to the standard output. Error messages will be printed to the console. When configuration errors are found CABM will print error messages, along with the associated files and line numbers where the errors were identified, for example,

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

4. The population section

4.1. Introduction

The population section (Section 4) describes the agent dynamics that occur within the spatial domain over time. They are follow the conventional population level dynamic labels such as death, birth, migration and growth but all occur at an agent level. For each process we will define the algorithm and mathematical representation and an example of the syntax for how you would incorporate each dynamic into your own model. This section also discusses how to define the temporal and spatial resolution, which are completely user defined albeit some are computationally limited.

The population section consists of several components, including:

- The spatial structure (Section 4.2);
- Annual Cycle;
- Model initialisation (i.e., the state of the partition at the start of the first; year);
- The years over which the model runs (i.e., the start and end years of the model)
- The annual cycle (time-steps and processes that are applied in each time-step);
- The specification and parameters of the population processes (i.e., processes that add, remove individuals to or from the partition, or shift numbers between ages areas);
- Selectivities;
- Layers (used by processes, observations and reports) and their definitions
- Parameter values and their definitions; and
- Derived quantities, required as parameters for some processes (e.g. mature biomass to resolve any density dependent processes, such as the spawner-recruit relationship in a recruitment process).

4.2. Spatial structure

The spatial structure of CABM is represented by a $n_{rows} \times n_{cols}$ grid, with rows $i = 1 \ldots n_{rows}$ and columns $j = 1 \ldots n_{cols}$. Each cell of this grid contains a set of agents denoted by $\boldsymbol{X_r} = \{X_{r,1}, \ldots, X_{r,n_r}\}$ where r denotes a cell containing n_r agents. Although these cells are discrete grids, agents can move be moved in a continuous manor where Cartesian points are stored by each agent over their existence.

CABM implements a single spatial domain, a grid of *square* cells (Figure 4.1). The spatial grid is completely user defined, but must be rectangular.

The dimensions of the spatial grid are user defined but must be at least a 1×1 grid (i.e., a single spatial cell), and the largest spatial structure currently allowed by CABM is a grid of 1000×1000 cells – although we note that models of this size are untested and will probably have very long run times.

Associated with the $n_{rows} \times n_{cols}$ spatial structure is the one compulsory layer (see Section 4.5), the *base layer*. This defines active cells in the grid, where an active cell can have agents whereas inactive cells cannot i.e., land. Active cells are defined in the base layer as values greater than zero. There

must be at least one cell in the spatial grid where agents can be present. In addition, the base layer also defines the relative *area* of each spatial cell that is used for density calculations within CABM.

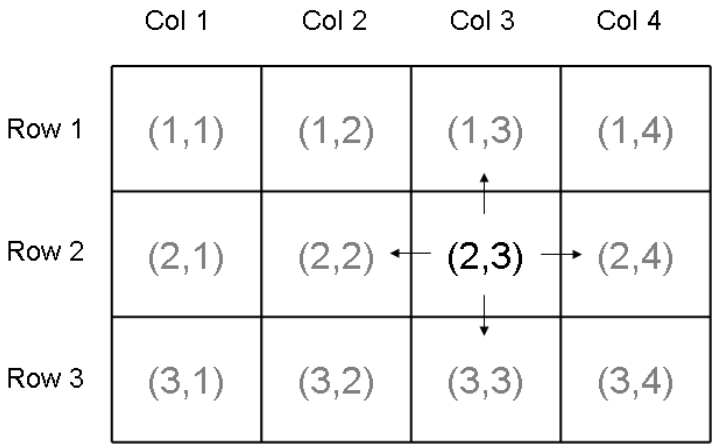


Figure 4.1: An illustration of the spatial structure

Models are implemented as a grid of square cells making up a rectangular matrix.

Hence, the definition of the spatial structure includes;

- The type of spatial grid and its dimensions, n_{rows} and n_{cols}
- The label of a numeric layer to be used as the base layer (defining the locations where the population can be present as well as the area of each cell)

For example, to specify a model with 3 rows and 4 columns (i.e., 12 spatial cells) with a base layer called `base`, then the syntax for `@model` would include,

```
@model
nrows 3
ncols 4
layer base

@layer base
type numeric
table layer
1 1 1
1 1 1
1 1 1
1 1 1
end_table
```

See Section 4.5 for how to define a layer using `@layer`. Some processes reports require the user to define latitude and longitude bounds for cells. As they are cell bounds the model expects one more bound than rows or columns. The format of latitude and longitude are 0-180 and 0-360, respectively.

If you wanted to add spatial areas where agents cannot exist i.e., land the base layer can be changed to include zeros such as in

```
@layer base
type numeric
table layer
0 1 1
0 1 1
0 0 1
1 1 1
end_table
```

4.3. Annual Cycle

The annual cycle defines which agent dynamics occur within a year and in what order. The annual cycle is made up of discrete time-steps, where each time-step has a set of observations to generate or dynamics to apply. The annual cycle is iteratively applied between `start_year` and `final_year`.

Each year is split into one or more time steps, with at least one process occurring in each time step. Each time step can be thought of as representing a particular part of the calendar year, or time steps can be treated as an abstract sequence of events. In every time step, there exists a mortality block: a group of consecutive mortality-based processes, where individuals are deleted from the population (see Section 4.6.2). This is an important concept to understand as derived quantities and some observations are associated with mortality blocks.

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

An example of syntax, to specify a model with the the above concepts is done using the `@model` block is specified as:

```
@model
start_year 1970
final_year 2018
min_age 1
max_age 20
nrows 2
ncols 4
age_plus_group True
initialisation_phases iterative
time_steps Summer Autumn Winter Spring

@time_step Summer
processes recruitment summer_fishery

@time_step Autumn
processes migrate_off_shore
...
```

4.4. Agents

CABM expresses a fish population as a collection of spatially referenced agents within a discrete spatial domain. Agents in spatial cell r are denoted by $\boldsymbol{X_r} = \{X_{r,1}, \ldots, X_{r,n_r}\}$, where n_r is the number of agents in cell r. Agent $X_{r,i}$ is also a set where each element in $X_{r,i}$ is an agent attribute outlined in Table 4.1. An agent represents n_i individual fish with homogeneous attributes. The number

of individuals an agent represents is based on the total abundance of the fish stock and number of agents used to represent that fish stock (Section 4.6.3). In general, the closer agents are to individuals ($n_i \to 1$) the longer model run times are. This leads to a trade off in the number of agents used to represent a fish stock and model run time.

Table 4.1: Attributes recorded for all agents, where $X_{r,i} = \{a_i, l_i, \ldots, x_i, y_i\}$.

Index	Description (variable type)
a_i	age (integer)
l_i	length (continuous)
lb_i	length bin (integer)
mat_i	maturity (boolean)
s_i	sex (boolean, 0 = male, 1 = female)
w_i	weight (continuous)
n_i	scalar = how many individuals this agent represents (continuous)
M_i	natural mortality rate (continuous)
$L_i^{\inf}$	growth parameters (continuous)
r_i^b	natal cell (birth cell)
r_i	current cell
x_i, y_i	coordinates over the domain (continuous)

Maturity is an attribute that is activated by having a maturity dynamic (see Section 4.7.2), it is tied to some specific derived quantities such as the type `mature_biomass`. You could also calculate the mature biomass on the fly, using the other derived quantities and selectivities, an example of this is shown in the maturity process section.

Sex is hard coded into the model and at this point the only way that sex effects model outcomes is through selectivities. So sex can have different maturation by age or length, and different vulnerability to mortality processes. The next additions are to have sex specific growth, which is a common occurrence in fish populations.

Lengths in the model are all defined by the length bins in on the `@model`, it is important to note that all calculations that are made via length are via the mid points of the bins supplied. Users supply lower and upper bins for the length classes and all calculations are based on the mid-points.

The other attributes don't need any explanations and are usually modified\used by processes, such as growth, movement and mortality.

4.5. Layers

Layers are a key underlying concept in CABM. They comprise of a grid of known values, with a value for every spatial cell in the model. Layers are used by processes, observations, and outputs commands to supply spatially explicit covariates and any categorical groupings required.

Layers are used by CABM to evaluate locations where the population may be present (via the *base layer*), to provide sets of known attributes or values of each spatial location (for some processes and for preference based movements), and to group or categorise cells for use by processes and observations. Layers consist of an $n_{rows} \times n_{cols}$ matrix and can be either *numeric* or *categorical*.

Every model must define at least one layer, the base layer L_B. A layer is defined as a $n_{rows} \times n_{cols}$ matrix of values (albeit with the exception of layers that describe distance — these are described in detail below), where the value in each cell represents a known quantity. For example layers may

represent classifications, physical attributes, or some other known or assumed quantity. Typically they are provided by the user as a matrix of values, although some layers (e.g., abundance or distance layers) can be calculated by CABM during a model run.

Within CABM, layers are used in the following contexts:

1. The base layer: The base layer L_B is a special layer (there must be exactly one base layer defined within the model) that defines the locations where agents can and cannot potentially be present (e.g., locations associated with the sea and not land in a marine model). Further, positive values of the base layer L_B represent the *area* represented by that spatial cell. Note, the values in the base layer must be numeric, but cannot be a meta-layer (*see below*).

2. Covariate layers: A model may have many covariate layers, and these are used as covariates of some population or movement process (e.g., the sea floor depth may be a covariate of some movement process). The values in layers used as covariates can be either numeric or categorical.

3. Classification layers: A model may have many classification layers, and these are used as a classification or grouping variable for aggregating data over individual spatial cells (i, j), e.g., statistical areas or management areas. Such layers are typically used to aggregate the population within cells into groups so-as to allow comparison with observations. The values in layers used as classification layers must be categorical.

CABM defines the following types of layer;

1. Numeric layers: A model may have many numeric layers, and these can be used as covariates of a population or movement process (e.g., depth may be a covariate of some movement process), and/or locations of event mortality. Numeric layers can contain only continuous (numeric) variables. Values for a numeric layer must be supplied for each cell by the user. Numeric layers can be rescaled to sum to some user-defined value, and unlike other layers, the values for each cell can be estimated.

   For example, to specify a numeric layer for a spatial model with 3×4 cells called, say `base`, with the top left and bottom right cells set to zero and all other cells set to one and not rescaled, use,

   ```
   @layer base
   type numeric
   data 0 1 1 1
   data 1 1 1 1
   data 1 1 1 0
   ```

2. Categorical layers: A model may have many categorical layers, and these can be used as a classification or grouping variable for aggregating data over individual cells, e.g., management areas; or as covariates of a population or movement process. Such layers are typically used to aggregate the population within cells into groups for comparing with observations, or to apply specific movement characteristics. The values in layers used as categorical layers can contain any characters (except white space), and are interpreted as categorical values. Values for a categorical layer must be supplied for each cell by the user.

   For example, to specify a categorical layer for a spatial model with 3×4 cells called, say `zone`, with the top left cells allocated as zone `A`, the bottom right allocated as zone `C`, and the rest as zone `B`, then use,

```
@layer zone
type categorical
data A A B B
data A A C C
data B B C C
```

3. Abundance layers: The abundance layer is the sum of the number of individuals within cell c with selectivity $S(a)$ at age a.

$$N(c) = \sum_{i \in \boldsymbol{X_c}} n_i I_i \tag{4.1}$$

$$I_i \sim Bern(S(a)) \tag{4.2}$$

with, I_i being either 1 that is the agent is calculated in the abundance calculation or 0 the agent is excluded from the calculation.

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

For example, to specify an abundance layer of all individuals who are categorised as `mature`, use,

```
@layer Abundance
type abundance
categories mature
selectivities One
```

4. Biomass layers: The biomass layer is the sum of the biomass of individuals within cell a in categories k, with selectivity S_l at age l, and mean weight w_{kl}

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

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

For example, to specify a biomass layer of all individuals who are categorised as `mature`, use,

```
@layer Biomass
type biomass
categories mature
selectivities One
```

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

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

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

For example, to specify an abundance density layer of all individuals who are categorised as `mature`, use,

```
@layer AbundanceDensity
type abundance_density
categories mature
selectivities One
```

6. Biomass-density layers: The biomass-density layer is the density of the biomass of individuals within cell a with area A_a in categories k, with selectivity S_l at age l, and mean weight w_{kl},

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

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

For example, to specify a biomass density layer of all individuals who are categorised as `mature`, use,

```
@layer BiomassDensity
type bioomass_density
categories mature
selectivities One
```

7. Meta-layers: CABM defines a special type of layer known as a *meta-layer*. Meta-layers allows individual layers to be indexed by year and applied as an annually varying layer within the model. For example, assume a model that uses Sea Surface Temperature (SST) as a layer, perhaps to drive some movement process. The SST values for each year of the model would be defined as individual layers, each with a unique label. A meta-layer could be defined that indexed the individual annual SST layers by year, and used as a covariate layer in the movement process. Meta-layers have a *default* layer that is used for time periods that are not specifically defined. Meta layers can be used wherever ordinary layers are used (except that they cannot be used as the base layer), with CABM extracting the appropriate layer value corresponding to the year or the initialisation phase.

   a) Numeric meta-layers: Numeric meta-layers are a meta layer of numeric layers — the individual ordinary layers that make up the meta-layer must all be of numeric type. For example, assuming that the layers `SST` and `SST1990`, `SST1995`, etc., are defined elsewhere, then to specify a numeric meta-layer with specific values for the years 1990-1995, and a default for all other years (including the initialisation phases), use,

```
@layer AnnualSST
type numeric_meta
default_layer SST
years    1990    1991    1992    1993    1994    1995
layers SST1990 SST1991 SST1992 SST1993 SST1994 SST1995
```

   b) Categorical meta-layers: Categorical meta-layers are a meta layer of categorical layers — the individual ordinary layers that make up the meta-layer must all be of categorical type. Categorical meta-layers are specified in the same way as numeric meta-layers. For example, assuming that the layers `zones` and `zone1990`, `zone1995`, etc., are defined elsewhere, then to specify a categorical meta-layer with specific values for the years 1990-1995, and a default for all other years (including the initialisation phases), use,

```
@layer AnnualCategoricalLayer
type categorical_meta
default_layer zones
years     1990     1991     1992     1993     1994     1995
layers zone1990 zone1991 zone1992 zone1993 zone1994 zone1995
```

4.6. Time sequences

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

- Annual cycle
- Mortality blocks
- Initialisation
- Model run years

4.6.1. Annual cycle

The annual cycle is implemented as a set of processes that occur, in a user-defined order, within each year. Time-steps are used to break the annual cycle into separate components, and allow observations to be associated with different time periods and processes. Any number of processes can occur within each time-step, in any order (although there are limitations around mortality based processes - see Section 4.6.2) and can occur multiple times within each time-step. Note that time-steps are not implemented during the initialisation phases (effectively, there is only one time-step), and that the annual cycle in the initialisation phases can, optionally, be different from that which is applied during the model years.

4.6.2. Mortality blocks

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

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

A mortality block is defined as a consecutive sequence of mortality processes within a time step. The processes that are mortality processes are all pre-defined in CABM, and cannot be modified. These mortality processes are described in subsection 4.7.4.

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

CABM will error out if more than one mortality block occurs in a single time step. The consequence of a mortality block is derived quantities and observations that wrap mortality blocks get executed before and after. This can be a costly exercise, if it is important that the model attributes accounts for some mortality then this is a cost that must be taken, but if, there will usually be a parameter that you could set to 0 or one and so CABM will only calculate the quantity once (either before or after). For example the parameter in the derived quantities class `proportion_through_mortality`.

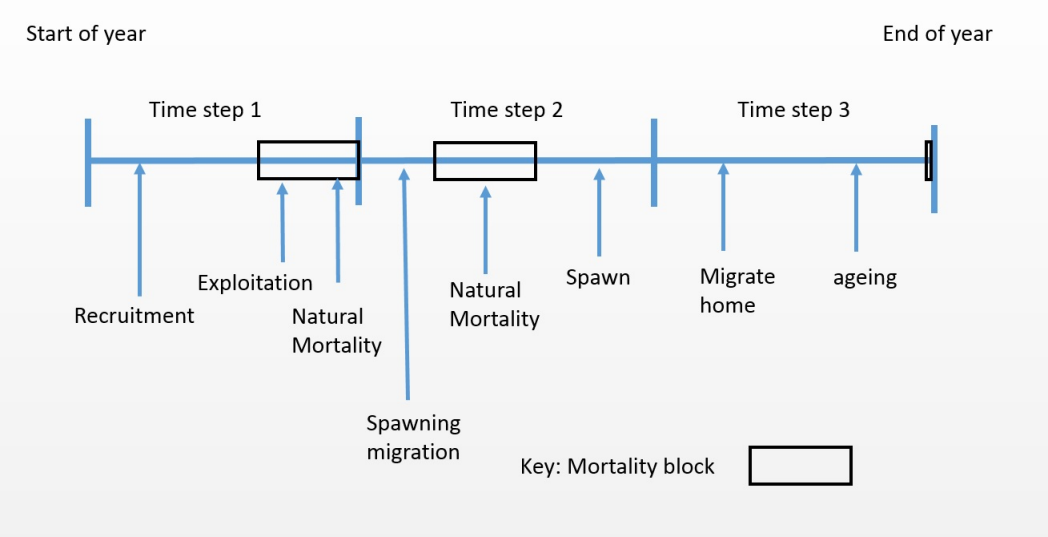


Figure 4.2: A visual representation of a hypothetical sequence for an annual cycle, with mortality blocks over multiple time steps.

4.6.3. Initialisation

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

CABM calculates the number of individuals that an agent represents during initialisation. It is derived following,

$$\widetilde{N} = \sum_{a=a_{min}}^{4\times a_{max}} R_0 e^{-M^* a} \ ,$$

$$\tilde{n} = \frac{\widetilde{N}}{n_{agents}} \ ,$$

where, a is the age, a_{min} is the minimum age, a_{max} is the maximum age, M^* is the initial natural mortality rate (it should be the same as $\bar{M}$ from the morality process, see Section 4.7.4), R_0 is the average number of individuals expected in the absence of fishing (see Section 4.7.3) and n_{agents} is the number of agents assumed from the users to model the initial stock. The choice of n_{agents} is a tradeoff between model run time and agent resolution of the stock. As n_{agents} increases CABM moves towards an IBM ($\tilde{n} \to 1$) but this comes at computational cost and larger model run times.

Once CABM calculates $\tilde{n}$, it creates the number of agents for the first age (a_{min}) in each cell denoted by $R_{r,a_{min}}$. This is calculated as

$$R_{r,a_{min}} = \lfloor R_0 / \tilde{n} P_r \rceil \ ,$$

where P_r is the proportion of initial population allocated to cell r and $\lfloor . \rceil$ denotes the rounding operator (the number of agents created needs to be an integer). CABM then creates the remaining number of agents for remaining ages in cell r following

$$R_{r,a} = \lfloor R_{r,a-1} \exp\{-M^*\} \rceil \ , \ a > a_{min} \ .$$

When agents are created, they are also assigned agent attributes based on their age and agent specific attributes. The above actions from CABM assume an equilibrium age-structure of agents in each

cell, but ignore movement and other dynamics that may effect starting conditions. To account for these dynamics, CABM then iterates over the annual cycle without fishing dynamics for a user defined number of cycles denoted by n_{init}. This populates the agents around the spatial domain according to the annual cycle assumptions.

The first thing CABM does is gets the parameter `number_of_agents` and spits that number of agents uniformly, over the spatial domain, alternatively the user could supply a layer `layer_label` (P_r) of proportions to seed the initial spatial distribution. This layer should sum to one so that the model initially seeds `number_of_agents`. When the CABM seeds the initial number of agents it also randomly assigns the agents an age based on an exponential distribution where the parameter λ of the exponential distribution is set by the command on the `@model`, `natural_mortality_process_label`. We suggest setting this command to the assumed natural mortality of the model. To see what age structure this would look like you can quickly use **R** to visualise. by running the following code in an **R** terminal you could see.

```
Z_param = 0.2;
agents_per_cell = 1000;
hist(rexp(agents_per_cell,Z_param), breaks = 30, xlab = "age", ylab = "frequency", main = "
    Initial age structure in each cell")
```

Once CABM has seeded the agents, it iterates over the annual cycle to change an approximated initialisation state to one that is more like what would occur for your annual cycle, this is controlled by the `years` command in the `@initialisation` block. The number of iterations in the iterative initialisation can effect the model output, and these should be chosen to be large enough to allow the population state to fully converge see Section 4.12 for some tips on initialising models.

Hence, for an iterative initialisation you need to define:

- The initialisation phases,
- The number of years in each phase
- the number of individuals that you want to represent the population
- The initial spatial distribution of seeded agents
- The stock\recruitment regions for setting the scalar.

Because the initialisation phase is responsible for seeding the initial agents, users must specify processes that the initialisation phase can seed parameters to agents. To see how parameters are set for each individual agent, users should see the individual processes in this Section 4.7.

An example of the syntax to implement this would be,

```
@model
...
initialisation_phases Iterative_initialisation

@initialisation_phase Iterative_initialisation
type iterative
initial_numbers_of_agents 1500000
years 50
layer_label initial_values_distribution
recruitment_layer_label recruitment_layer
```

For an example of setting an initialisation block with multiple stocks see the 3area example.

4.7. Agent dynamics

Agent dynamics are functions responsible for modifying agents (growth), moving agents, creating agents (recruitment) and deleting agents (mortality). The inputs for these functions use both agent specific attributes (Table 4.1) and assumed parameter values. Agent dynamics are predominately stochastic, where agent actions are based on a randomly generated realisation from a random variable. For example, a dynamic that results in a binary outcome for an agent (i.e. an agent getting caught by a fishing interaction or an agent maturing) can be expressed by the Bernoulli random variable $I \sim \mathcal{B}ern(p)$, where p is the probability of an event occurring

$$I \begin{cases} 1, & \text{event occurs} \\ 0, & \text{event does not occur} \end{cases} .$$

Agent dynamics generally iterate over a collection of agents and for each agent, they apply an outcome based on a realisation from a random variable. The following example demonstrates the notation used when describing how an agent dynamic applies an event to all agents in cell r,

$$\begin{aligned} p_i &= f(X_{r,i}, \boldsymbol{\theta}) \ , \ \forall \, X_{r,i} \in \boldsymbol{X_r} \ , \\ I_i &\sim \mathcal{B}ern(p_i) \\ I_i &\begin{cases} 1, & \text{event occurs for the } i^{th} \text{ agent} \\ 0, & \text{event does not occur for the } i^{th} \text{ agent} \end{cases} , \end{aligned}$$

where p_i denotes the i^{th} agents ($X_{r,i}$) specific probability of the event occurring. This is the result of the function $f(.)$ which uses agent attributes and parameters $\boldsymbol{\theta}$.

The agent dynamics described in the following sections define p_i as a function of specific agent attributes (Table 4.1). In all descriptions $X_{r,i}$ in $f(X_{r,i}, \boldsymbol{\theta})$ is replaced with the specific agent attribute. For example, if p_i was a function of an agent's length (l_i), it would be defined as,

$$p_i = f(l_i, \boldsymbol{\theta}) \ , \ \forall \, X_{r,i} \in \boldsymbol{X_r} \ .$$

4.7.1. Ageing

Ageing is an implicit process in the model, Each agent that is created or recruited gets assigned a birth year. This means that when ever we want to ask for the agent we just calculate `current_year - birth_year`, thus there is no explicit ageing process. Note that we do return the `max_age` of the model, if the agent is older than that age (so we only work with the truncated age distribution).

This means every fish automatically ages by one at the end of the year, or you could think of ageing a fish at the very beginning of the year (tomayto, tomahto), just be aware that you don't have control over this process. Something to keep in mind is that growth is not directly tied with ageing, that is a individual that ages one year does not also get a years worth of growth. This unlike most population based models that will usually have an implicit assumption of growth with ageing. Although if one puts a growth process at the beginning or end of an annual cycle this can be achieved. This is just a side note to keep in mind when trying to align this model with others you may want to test against.

4.7.2. Maturity

The process of converting an agent from immature to mature, only should be used if you link it to a mature-biomass derived quantity (Section 4.8). This process only changes the internal state of an agent i.e. no material change happens to the agent. This could be useful down the track if we have dynamics that only occur to mature agents and so this internal flag can be checked against for use in a different algorithm. Maturity is applied using the ogive defined in the subcommand `maturity_ogive_label` in the `@model` block, and is applied for a single agent (assuming the agent is not already mature)

Maturity is applied to agents in cell r at time-step t following

$$
\begin{aligned}
&P_i = S(a_i), \quad \forall\, X_{r,i} \in \boldsymbol{X_r} \ , \\
&I_i \sim \mathcal{B}ern(1 - P_i) \ , \\
&I_i \begin{cases} 1, & \text{agent becomes mature} \\ 0, & \text{agent remains immature} \end{cases} ,
\end{aligned}
$$

where, P_a is the probability of an immature agent at age a_i becoming a mature agent, defined by the selectivity $S(.)$. Currently maturity can only be an age based process, although this is not difficult to make a length based probability statement if someone in the future wanted this. A note about creating a logistic based maturity schedule. We recommend users use the selectivity `type = logistic_producing` (Section 4.9) or a similar style of selectivity. The results will not be as expected if you use a normal logistic selectivity. This is because this is only applied to the non-mature agents and so the selectivity needs more thought than other situations.

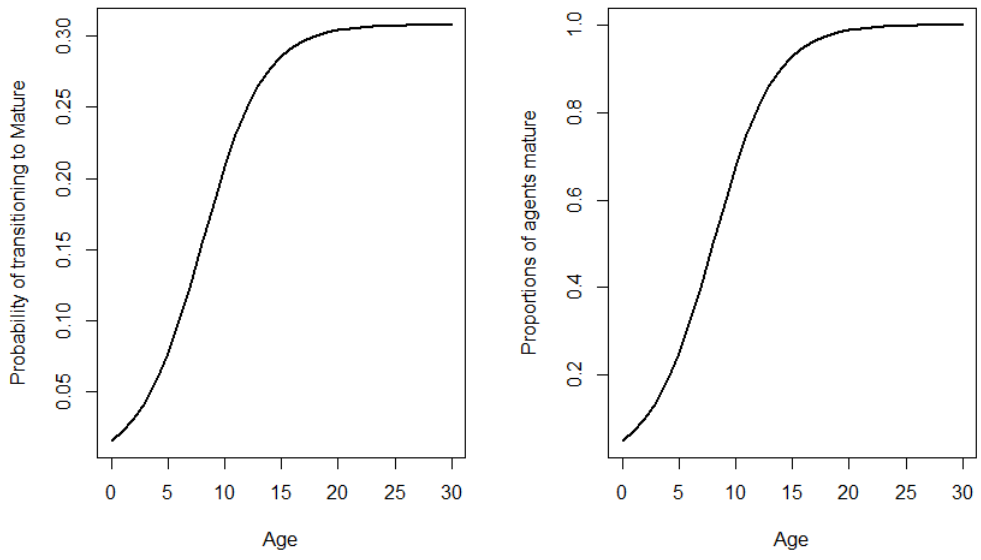


Figure 4.3: A comparison of maturity schedules, on the left we have used a logistic producing ogive, and on the right if we apply the maturity ogive over and over you get the resulting proportion mature in the right hand panel (which is logistic).

An alternative way and perhaps more efficient method for capturing a snap shot of the mature biomass or abundance of all individuals in the population is by using the generic biomass or

abundance derived quantities (Section 4.8 4.8). Where we calculate what is mature at the time of the derived quantity, rather than having an explicit process that allows users to separate the maturation process from the summarising of abundance of biomass of the mature component. Examples of how you would set up the two methods in CABM are below, firstly with maturity as an explicit process and secondly as it being implicit.

```
@model
...
maturity_ogive_label mature_ogive # define maturity selectivity: male female (order is important!!)

@time_step Annual
processes  Maturation

@time_step Spawn
processes  no_process

@process Maturation  # This will use the maturity ogive on the @model
type maturation

@derived_quantitiy SSB  # get a snap shot of mature biomass
type mature_biomass
time_step Spawn
biomass_layer_label SSB_areas
proportion_through_mortality_block 0.0
```

The above model setup will have agents in the system that are mature and immature and we can call the specialised `mature_biomass` derived quantity to get a snap shot at any point in the system to view the biomass or abundance. The other way of dealing with maturity is implicitly as shown below.

```
@model
...
maturity_ogive_label One // define a constant selectivity here, just to keep the model happy

@time_step Annual
processes  Other_processes

@time_step Spawn
processes  no_process

@derived_quantitiy SSB  // get a snap shot of mature biomass
type biomass
time_step Spawn
selectivity mature_ogive  // Define mature selectivity here
biomass_layer_label SSB_areas
proportion_through_mortality_block 0.0
```

So just to re-iterate the difference, you can have an explicit maturing process that is not associated with a derived quantity, or you could purely associate a derived quantity to the maturing process. This brings up a useful point in that, given the generality of the program there are more than one way to approximate dynamics and some will be more computationally quicker than others. So have a play around but try and keep the number of times we iterate over the state as small as possible, as you will get significant time saving.

4.7.3. Recruitment

Recruitment is the process where new agents are created in the system. It is also the process that defines stock structure in the model. Stocks are not an explicit attribute of CABM so consideration on this process is important. How the stock is defined using the recruitment dynamic surrounds where we calculate Spawning Stock Biomass (*SSB*) and where the resulting recruits are first seeded. This means that for all recruitment types you will need to specify a B_0 or R_0 parameter and an associated derived quantity that defines the spatial resolution of the stock (*SSB*) for that event. This is how an R_0 value is calculated from the initialisation phase.

$$X = \sum_{a=0}^{4A} exp^{-Ma} \tag{4.6}$$

Where, M is the average natural mortality rate as defined by the `natural_mortality_process_label` in the `@model` block. A is the maximum age of the population, 4 is an arbitrary scalar to make sure that we cover the entire age distribution. Once we have our multiplier (X), we then

$$R_{0,s} = \frac{N_a}{X} \frac{B_{0,s}}{\sum_s B_{0,s}} \tag{4.7}$$

where, $R_{0,s}$ is the stock specific initial/equilibrium number of agents to seed, N_a is the number of agents in the model world defined in the `@initialisation_phase` using the subcommand `number_of_individuals` and $B_{0,s}$ is the user defined stock specific equilibrium spawning stock biomass.

4.7.3.1. Constant recruitment

$$R_{y,s} = R_{0,s} \tag{4.8}$$

```
# Constant
@process Recruitment_const
type recruitment_constant
recruitment_layer_label recruit_layer
b0 29924.6
ssb SSB

@derived_quantity SSB
type mature_biomass
biomass_layer_label SSB_areas
time_step Annual
proportion_through_mortality_block 0.5

# proportion of where recruits are spatially distributed
@layer recruit_layer
type numeric
table layer
0.2 0.2 0.2
0.2 0.2 0
0 0 0
0 0 0
```

```
end_table

# define areas where SSB calculations are made
@layer SSB_areas
type numeric
table layer
1 1 1
1 1 0
0 0 0
0 0 0
end_table
```

4.7.3.2. Beverton-Holt recruitment

We have taken the parametrisation akin to Mace and Doonan (1988) that is a population based formula. There is room to make this a more pure agent based model where we look at probability of finding a mate and etc, but for know this is all we have.

$$R_{y,s} = R_{0,s} \times \frac{SSB_{y-ssboffset,s}}{B_{0,s}} / \left(1 - \frac{5h-1}{4h}\left(1 - \frac{SSB_{y-ssboffset,s}}{B_{0,s}}\right)\right) \times YCS_y \tag{4.9}$$

where h is the compensation parameter known as the steepness parameter that represents the number of agents when the stock SSB is at 20% of B_0, YCS_y is the year class strength parameter also termed the stock recruitment residual that accounts for any deviation of the deterministic Beverton Holt relationship due to things like predation, environment etc. If there are multiple spatial cells that are associated with a recruitment event then the allocation to a single cell is a simple multiplication with a proportion e.g.

$$R_{y,s,c} = R_{y,s} \times p_c \tag{4.10}$$

where $R_{y,s,c}$ is the resulting agents in cell c with proportion p_c there is no stochastic behaviour in this process unlike other processes.

```
# Recruitment Beverton Holt
@process Recruitment
type recruitment_beverton_holt
recruitment_layer_label recruit_layer
b0 100000
steepness 0.85
ycs_values 1*53
ssb SSB
```

4.7.4. Mortality

Mortality processes remove agents from the model domain, and do not allow them to be available to processes or summaries. Mortality that are associated with fishing or anthropogenic removals are also responsible for tag-recaptures.

Mortality types that can be used to search for tag-recaptures are, `mortality_event_biomass`, `mortality_baranov` and `mortality_effort_based`. I will describe how this is handled in CABM here as the method is the same through out all three mortality processes. The inputs that will

control how we sample for tag-recaptures are `scanning_years` and `scanning_proportion`. These basically just say what years we are scan for tags and what proportion of the F-mortality do we check for tags. One thing to note that has been plaguing me a little bit is the recapture probability. If you have an agent based model where an agent represents multiple individuals and you have tagging process (sub-section 4.7.7) which splits out a true individual. CABM does not change the probability statements on how many individuals an agent represents i.e. all agents have equal probability. This means we may have the same probability of recapturing an agent that represents 10000 fish as we do an agent that represents a single individual. I haven't come up with a clever way around this, so this is purely a reminder to users to consider this before going crazy on any analysis. To extract information on tag-recaptures you will have to generate a report for the specific mortality process, as well as that you will have to specify the subcommand `print_extra_info` in the `@process` mortality block.

4.7.4.1. Constant mortality rate

Apply an instantaneous mortality rate, and we remove the individual if the following condition is met which compares the survivorship ($1 - e^{-M_i * S_a}$) with a draw from a uniform random variable ($U()$).

$$\text{if } U() \leq 1 - e^{-M_i * S_a} \text{ agent dies} \tag{4.11}$$

where, M_i is the individuals instantaneous mortality rate and S_a is a selectivity at age allowing the process to be age based (but can be length based). M_i the individuals instantaneous mortality rate is assigned from generating a random value from either a normal or lognormal distribution by using the subcommand `distribution`

$$M_i \sim N(M * M_{row,col}, CV \times M * M_{row,col}) \text{ with syntax } \sim N(\mu, \sigma) \tag{4.12}$$

or

$$M_i \sim LN(M * M_{row,col}, CV) \text{ with syntax } \sim LN(E[X], CV) \tag{4.13}$$

where M is the parameter `m` and $M_{row,col}$ is the multiplier layer value in row *row* and column *col*. For the lognormal parametrisation we make users define the expectation rather than the μ parameter as it seems more intuitive and this follows that $E[X] = e^{\mu + 0.5\sigma^2}$ Users can also set a flag (`update_mortality_parameters`) that says if individuals move they will take on a new mortality rate parameter of the new area, if not they will retain their mortality rate assigned to them at birth.

An example of how you could set up a constant mortality rate process that varied through time, space and by age is below

```
@process natural_mortality
type mortality_constant_rate
update_mortality_parameters true
m_multiplier_layer_label M_layer
m 0.15
distribution lognormal
```

```
cv 0.1
selectivity_label natural_mortality_by_age

@layer M_layer
type numeric
table layer
1 1.2 1.1    # 0.15 0.18 0.165
1 0.9 0.8    # 0.15 0.135 0.12
0.8 0.8 0.7  # 0.12 0.12 0.105
end_table

@selectivity natural_mortality_by_age
type double_exponential
x1 0
x0 6
x2 30
y1 0.6
y0 0.1
y2 0.4

@time_varying m_by_year
type constant
parameter process[natural_mortality].m
values 0.134 0.153 0.106 0.1833
years 1990 1991 1992 1993
```

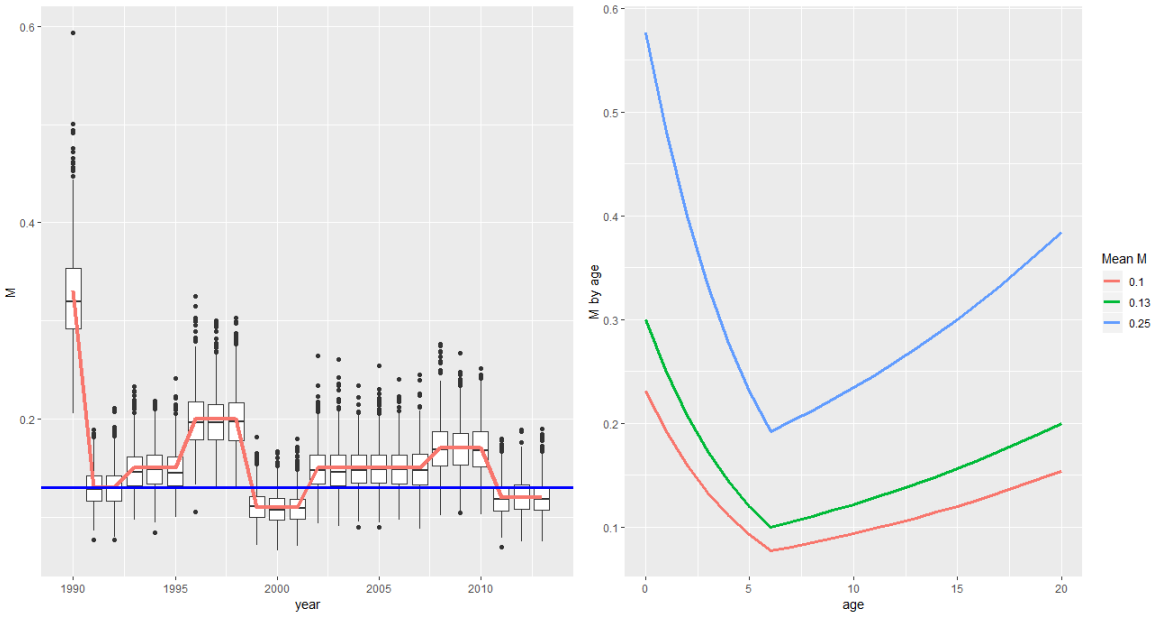


Figure 4.4: The left plot show how uses can vary the Mean M by year, with the blue line being the value used to initialise the agents, the red value is the input and the box and whisker's represent the actual M assigned to each agent from a random sample of 1000. In the right plot we see how using selectivities the user can also have age varying M, here we use a double exponential curve which is demonstrated with a range of mean M values that could occur in a year.

4.7.4.2. Event-Biomass

This is a removal event such as fishing, where we iterate over a range of cells and remove agents based on some selectivity or spatial vulnerability. This is the simplest way to remove individuals from an anthropogenic process, however it is quite annoying to set up for a spatial model with a fine spatial resolution as you need to specify a catch for each cell. An example of how this process is set up is specified below, if you want an alternative fishing process that is based in theoretiical spatial fishing distributions see the next section (Section 4.7.4.7). Advantages using this mortality process say over the Baranov method (next paragraph **??**) is that it allows for agents to have multiple encounters with a fishery and it can deal with multiple fisheries with different selectivities and spatial patterns, but the negatives is that you cannot simply incorporate simultaneously natural mortality processes, and may be more suited for a pulse like fishery.

The algorithm

- randomly select an agent, **with replacement**
- randomly select a fishery (if more than one), **with replacement**
- check to see if this agent is vulnerable to the fishery either age or length based process, by comparing a uniform random draw with a ogive
- if we are scanning fish for tag-recaptures check for tags
- if we have minimum legal size (MLS) check to see if the fish will be returned this is a true or false statement, if agent length $<$ MLS return with some handling mortality probability that will be again checked with a uniform random variable
- if not MLS check then we remove this agent from the population and add it to the catch (all agents are removed, this must be remembered)
- repeat the above steps until we have removed a desired biomass or if there are not enough fish to remove we exit with an error.

The key here is that an agent can interact with fishery many times during a fishing event, but there is no natural mortality.

```
@process Fishing_Mortality
type mortality_event_biomass
print_extra_info true ## this may print too much information and slow down model
table catch ## these are labels for @layer that describe catch over space.
year FishingTrwl FishingLine
1972 1972_Trwl 1972_Line
1973 1973_Trwl 1973_Line
1974 1974_Trwl 1974_Line
end_table

table scanning ## proportion of catch scanned for tags
year FishingTrwl FishingLine
1972 0.3 0.5
1973 0.6 0.7
1974 0.2 0.4
end_table

## describe fishery specific parameters
table method_info
method          male_selectivity    female_selectivity  minimum_legel_length  handling_mortality
FishingTrwl     trwlFSel_m          trwlFSel_f 0 0
FishingLine     Sel_Line            Sel_Line 0 0
end_table
## if single sex model only have one column called selectivity
table method_info
method        selectivity  minimum_legel_length handling_mortality
FishingTrwl   trwlFSel     0                    0
FishingLine   Sel_Line     0                    0
end_table
```

4.7.4.3. Baranov Simple

This process applies spatially explicit fishing mortality for only a single fishery and natural mortality. See section 4.7.4.5 if you want to consider multiple fisheries.

The algorithm

- iterate over every individual in the partition or spatial cell
- See if the fish dies by comparing total mortality survivor ship $U() \leq 1 - e^{-M_a + F_a}$ assuming M and F are both age based, but they can be length based (M_l, F_l) or a combination
- If fish dies see if it was from M_a or F_a if ($U() < \frac{F_a}{F_a + M_a}$) then it was removed from an fishing event and is included in fishing specific observations.
- we also can apply MLS and handling survivor ship
- we can also choose to check for tag-recaptures

So from the algorithm above we can see that agents are exposed to both M and F, however the negative is that this assumes only a single interaction with the fishery which is an unlikely assumption. However I added this process because it both takes M and F simultaneously and may actually be faster than an event biomass dynamic (previous paragraph 4.7.4.2).

4.7.4.4. Exploitation

This is a removal event where we iterate over a range of cells and remove agents based on multiple fisheries and spatially explicit fishing mortality rates. Advantages using this mortality process over the event biomass event (previous paragraph **??**) is that it is alot faster (computationally) another benefit is that it allows for a constant exploitation pattern, even under different recruitment trends, unlike a catch based mortality event.

for multiple fisheries, in any given cell we calculate the total exploitation rate as

$$U_b = \sum_{f=1} U_f S_b^f \tag{4.14}$$

$$p_b^f = \frac{U_f S_b^f}{F_b} \tag{4.15}$$

where, f indexes the fishery, and S_b^f is the selectivity for bin b (can be either age or length) for fishery f and U_f is the fishing exploitation rate in the cell we are in. p_b^f is the proportion of fishing mortality for bin b attributed ot fishery f.
The algorithm follows

- Iterate over each agent that is alive and see if agent i is caught ($U() < 1 - e^{F_{b_i}}$) with b_i is the bin of agent i
- If caught see which fishery caught it using a $f \sim multinomial(p_1^f, \ldots p_{n_f}^f)$
- if we have minimum legal size (MLS) check to see if the fish will be returned this is a true or false statement, if agent length $<$ MLS return with some handling mortality probability that will be again checked with a uniform random variable
- if not MLS check then we remove this agent from the population and add it to the catch (all agents are removed, this must be remembered)

The key here is that an agent can interact with fishery many times during the process execution, but there is no natural mortality.

```
@process Fishing_Mortality
type mortality_exploitation
print_extra_info true ## this may print too much information and slow down table f_table
table u_table
## these are labels for @layer that describe U over space for each fishery
year FishingTrwl FishingLine
1972 1972_Trwl 1972_Line
1973 1973_Trwl 1973_Line
1974 1974_Trwl 1974_Line
end_table

## describe fishery specific parameters
table method_info
method          male_selectivity    female_selectivity  minimum_legel_length  handling_mortality
FishingTrwl     trwlFSel_m          trwlFSel_f 0 0
FishingLine     Sel_Line            Sel_Line 0 0
end_table
```

4.7.4.5. Baranov

This is a removal event where we iterate over a range of cells and remove agents based on multiple fisheries and spatially explicit fishing mortality rates. Advantages using this mortality process over the event biomass event (previous paragraph **??**) is that it is alot faster (computationally) another benefit is that it allows for a constant exploitation pattern, even under different recruitment trends, unlike a catch based mortality event.

for multiple fisheries, in any given cell we calculate the total fishing mortality as

$$F_b = \sum_{f=1} F_f S_b^f \tag{4.16}$$

$$p_b^f = \frac{F_f S_b^f}{F_b} \tag{4.17}$$

where, f indexes the fishery, and S_b^f is the selectivity for bin b (can be either age or length) for fishery f and F_f is the fishing mortality rate in the cell we are in. p_b^f is the proportion of fishing mortality for bin b attributed ot fishery f.
The algorithm follows

- Iterate over each agent that is alive and see if agent i is caught ($U() < 1 - e^{F_{b_i}}$) with b_i is the bin of agent i
- If caught see which fishery caught it using a $f \sim multinomial(p_1^f, \ldots p_{n_f}^f)$
- if we have minimum legal size (MLS) check to see if the fish will be returned this is a true or false statement, if agent length $<$ MLS return with some handling mortality probability that will be again checked with a uniform random variable
- if not MLS check then we remove this agent from the population and add it to the catch (all agents are removed, this must be remembered)

The key here is that an agent can interact with fishery many times during the process execution, but there is no natural mortality.

```
@process Fishing_Mortality
type mortality_baranov
print_extra_info true ## this may print too much information and slow down table f_table
table f_table
 ## these are labels for @layer that describe F over space for each fishery
year FishingTrwl FishingLine
1972 1972_Trwl 1972_Line
1973 1973_Trwl 1973_Line
1974 1974_Trwl 1974_Line
end_table

## describe fishery specific parameters
table method_info
method          male_selectivity    female_selectivity  minimum_legel_length  handling_mortality
FishingTrwl     trwlFSel_m          trwlFSel_f 0 0
FishingLine     Sel_Line            Sel_Line 0 0
end_table
## if single sex model only have one column called selectivity
table method_info
method        selectivity  minimum_legel_length handling_mortality
FishingTrwl   trwlFSel     0                    0
FishingLine   Sel_Line     0                    0
end_table
```

4.7.4.6. Hybrid

Note: this process should only run for run mode `ibm -m`. This will apply Baranov mortality process (section 4.7.4.5) between `start_year` and the fisrt year in `assessment_years` subcomand then applies a catch based mortality `mortality_event_biomass` (section 4.7.4.2) over the remaining assessment years.

The input is identical to Baranov mortality process (section 4.7.4.5) where users should only supply layer labels in table `f_table` catch is determined by the **R** interaction.

4.7.4.7. Effort-Based F

A lot of this process was inspired by Truesdell et al. (2017), where users can specify some theoretical fishing distribution, currently only Ideal Free Distribution is allowed (but I am hoping to add a weighted ideal free distribution with user defined weights) and a total catch. CABM applies this process by calculating the vulnerable biomass over all spatial cells ($V_{row,col}$), where each individual contributing to the vulnerable biomass is done deterministically **unlike** most other systems in the model (the stochastic nature comes later Equation 4.19),

$$V_{row,col} = \sum_{i \in (row,col)} \bar{w}_i \times s_i \times P_c \tag{4.18}$$

where, P_c is the probability of capture which can be length or age based, i indexes all the available agents in cell row, col, and $\bar{w}_i$ s_i are the weight and scalar of the agent respectively, ()I is the identity function based on the outcome of the Bernoulli trial. Once we have the vulnerable biomass by cell we want to allocate our theoretical effort. The user can specify effort two ways. The user can specify a vector of expected effort one for each enabled cell, this would mean that effort was time-invariant. The second is that user could specify a numeric layer of efforts for each year (the order

is not important because CABM will align them based on the theoretical distribution). **Note** units of effort could be important for solving λ look at the end of this section to see technical details on this. Once we have effort ($E_{row,col}$) and vulnerable ($V_{row,col}$), the CABM will align the highest effort value with the cell with the highest biomass, for the ideal free distribution. Once we have effort and vulnerable biomass we need to estimate a parameter λ_y so that we solve for catch.

$$\hat{C}_{y,row,col} = \sum_{i \in (row,col)} (U() \leq 1 - e^{\lambda_y E_{row,col} P_c}) \mathrm{I}\ \bar{w}_i \times s_i \tag{4.19}$$

A note from the above equation the uniform random draw $U()$ is set once for all agents and is not re-drawn every time we iterate over this equation to minimise the Equation 4.21 below to solve for λ_y.

$$\hat{C}_y = \sum_{row} \sum_{col} \hat{C}_{y,row,col} \tag{4.20}$$

We then minimiser the sum of square of the following expression to solve for λ_y,

$$\hat{\lambda}_y = argmin_{\lambda} (C_y - \hat{C}_y)^2 \tag{4.21}$$

where, C_y is the user catch, in this case $\hat{F}_y = \hat{\lambda}_y E_{row,col}$. We finally removals with our estimated F using the normal survivor ship process, Bernoulli trial if $U() \leq 1 - e^{\hat{F}_y P_c}$ then the agent dies.

A technical note to consider, is that the λ parameter is assumed to $\lambda \in (0.0000001, 100)$. So you will need to scale and check your Effort units so that, with the above constraint and your effort we can solve for the Catch in that year. Also there is an option to specify the starting value when solving for λ using the subcommand `starting_value_for_lambda`. If you print this process it should tell you how long it takes in seconds to solve for lambda for each year. See Section 5.1 for information on the minimiser used in the optimisation routine.

```
@process summer_fishery
type mortality_effort_based
years 1991
catches 2234
selectivity trawl_selectivity
minimiser minimiser_for_summer_fishery
effort_values 0.076 0.089 0.219 0.104 ## for 4 cell model


@minimiser minimiser_for_summer_fishery
type numerical_differences
## play with these if you have trouble minimising or starting value
iterations 20
evaluations 30
tolerance 0.03
step_size 0.003
```

4.7.4.8. Effort-Based F with covariates

This process leads on from Effort-Based F above but allows for layers to control effort distribution other than vulnerable biomass. For example economic-proxy layers such as distance from the coast (Figure 4.5)

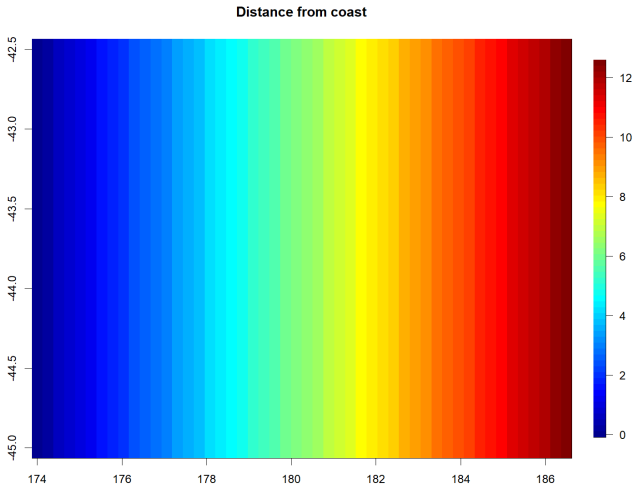


Figure 4.5: A layer that represents distance from the eastern coast line

Given a numeric layer, you need to assume a preference function (Section 4.10) which represents the likely areas for effort e.g Figure 4.6

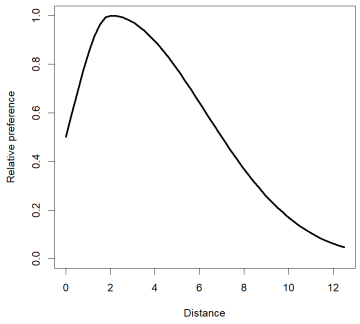


Figure 4.6: Preference function

These effort covariates also have an associated weight (multiplier) which is defined by the subcommand `preference_weights`. The resulting relative spatial effort distribution denoted by $\tilde{E}_{ij}$ is the arithmetic mean of these covariates and the underlying vulnerable biomass.

$$\tilde{E}_{ij} = \frac{1}{n_k + 1} \sum_{k=1}^{n_k} P_{ij}^{k} w_k \frac{V_{ij}}{\sum_i \sum_j V_{ij}} \tag{4.22}$$

with,

- V_{ij} being vulnerable biomass in cell ij

- n_k number of covariates
- P_{ij}^k preference value for covariate k in cell ij
- w_k `preference_weights` for covariate k
- V_{ij} being vulnerable biomass in cell ij
- V_{ij} being vulnerable biomass in cell ij

What this can look like, is take the disatnce from the coast example (Taken from the Example model PreferenceMovementSpatialFishing) if in a given year the vulnerable biomass was distributed as shown Figure 4.7

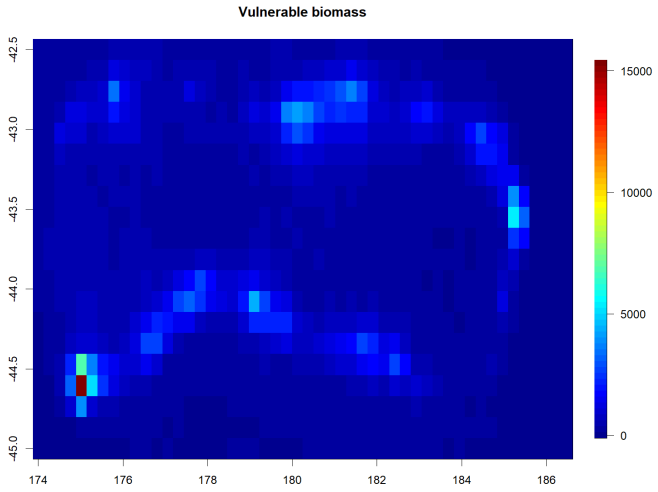


Figure 4.7: Vulnerable biomass at the time of fishing

assuming the `preference_weights` for distance is 0.5 the resulting relative spatial effort distribution ($\tilde{E}_{ij}$) would look like Figure 4.8

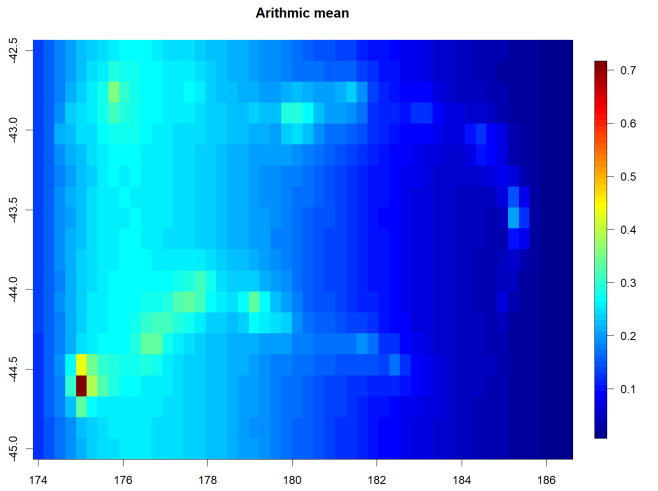


Figure 4.8: Vulnerable biomass at the time of fishing

the effort_values supplied by the user are then distributed proportional to $\tilde{E}_{ij}$ as in the previous effort based approach (Section 4.7.4.7). The algorithm then follows exactly the same as Section 4.7.4.7, with the minimisation. To configure this process the following syntax should work.

```
@process summer_fishery
type mortality_effort_with_covariates
years 1991
catches 2234
selectivity trawl_selectivity
minimiser minimiser_for_summer_fishery
effort_values 0.076 0.089 0.219 0.104 ## for 4 cell model
preference_functions dist_pref
preference_layers distance_from_shore
preference_weights 0.5

@minimiser minimiser_for_summer_fishery
type numerical_differences
## play with these if you have trouble minimising or starting value
iterations 20
evaluations 30
tolerance 0.03
step_size 0.003

## Preference Functions
@preference_function dist_pref
type double_normal
mu 2
sigma_l 2
sigma_r 5

@layer distance_from_shore
type numeric
table layer
0 0.255102 0.5102041 0.7653061
0 0.255102 0.5102041 0.7653061
0 0.255102 0.5102041 0.7653061
0 0.255102 0.5102041 0.7653061
end_table
```

4.7.4.9. Mortality Cull

Not really a realistic/common mortality event, but useful when looking at movement dynamics. This mortality process will kill all agents in the specified cells.

```
@process summer_fishery
type mortality_effort_based
years 1991
layer_label kill_all_agents

@layer kill_all_agents
type integer
table layer
0 0 0
1 1 1
end_table
```

This process will kill all agents in the bottom cells. Any cell with a value ¿ 0, all agents will be removed.

4.7.5. Movement

4.7.5.1. Box Transfer

The box transfer movement process is a simple movement process that is commonly applied in stock assessments. It describes a proportion of individuals moving from a cell to all other cells. This is applied by drawing from a multinomial distribution where an individual will end up. There are two types of movement that users can select for this method; `markovian` and `natal_homing`. The difference is with markovian movement the `origin_cell` refers to the cell an individual is currently in, where as if the movement type is natal homing the `origin_cell` refers to the cell that the individual was born in. These are subtle in application but a quite different movement assumptions. A practical note about the two methods natal homing is difficult to thread so may be quite a lot slower (disclaimer I haven't tested it, but I have turned off threading in this process). The probability is not age or length specific and so applies equally to all individuals in a cell. An example of syntax of how you would set this process up for a 3 area model is below.

```
@process Movement
type movement_box_transfer
origin_cell 1-1 2-1 3-1
probability_layers move_from_cell_1 move_from_cell_2 move_from_cell_3
movement_type markovian

@layer move_from_cell_1
type numeric
proportions true
table layer
0.3 0.2 0.5
end_table

@layer move_from_cell_2
type numeric
proportions true
table layer
0.1 0.6 0.3
end_table

@layer move_from_cell_3
type numeric
proportions true
table layer
0.55 0.25 0.2
end_table
```

The above assumes that the probability of movement from one cell to another doesn't change over time. Time varying movement can be applied by using the numeric meta layers see Section 4.5 for more information. A quick example of how you can apply this following the above example

```
@layer move_from_cell_1
type numeric_meta
```

```
years 1990:2000
layer_labels pre_2000_movement post_2000_movement
default_layer pre_2000_movement

@layer pre_2000_movement
type numeric
proportions true
table layer
0.1 0.6 0.3
end_table

@layer post_2000_movement
type numeric
proportions true
table layer
0.55 0.25 0.2
end_table
```

You can see from the above example that the movement from cell 1 to all other cells has different probabilities from before 2000 and after 2000. The above example could be generalised furthur having a different probability matrix for every year, I kept it simple for illustration.

4.7.5.2. Preference Based movement

The preference in cell i for preference variable x can be described as,

$$P_i^x = f(x_i;\theta) \tag{4.23}$$

where $f(x_i;\theta)$ is a preference function (see Section 4.10). The overall preference for cell i is the geometric mean of all preference functions from all layers.

$$P_i = \left(\prod_n^{i=1} P_i^x\right)^{1/n} \tag{4.24}$$

where n is the number of preference layers that are involved in the movement process. Once we have the preference for all cells in the spatial domain, we need to calculate the gradient or velocity field in both the **X** (E-W) and **Y** (N-S) axis. Currently CABM uses the forward, backwards, and central difference approximation. This obviously sucks as coarse resolution but as we get fine scale models the approximation more accurately will describe the change in preference in each direction. The central approximation has been used in other habitat based preference calculations (Phillips et al., 2018) and is applied like this

$$u_i = \frac{P_{j,i+h} - P_{j,i-h}}{2} \tag{4.25}$$

where h is some distance or neighbour cell, this yields us gradient or velocity in both directions then we use a bias random walk to move individuals in space.

$$x_j \sim N(u_i, \sigma_i) \tag{4.26}$$

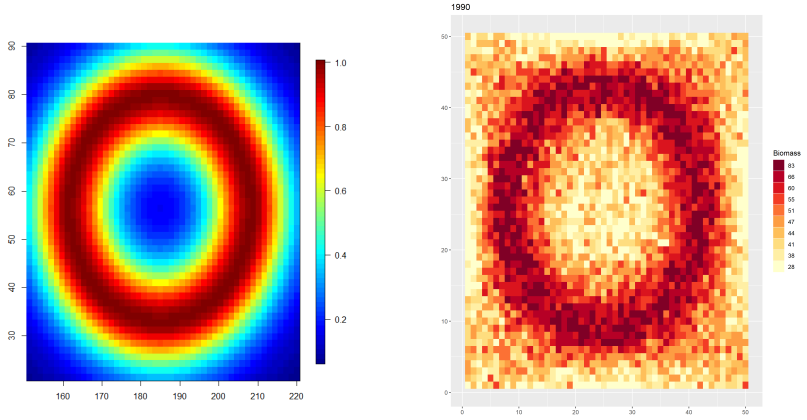
Where u_i is the gradient in the **X** direction, D_i is the diffusion parameter that is scaled based on the habitat preference and x_j is the future location on the X axis of the individual. D_i inversely relates to the quality of the preference habitat and a parameter D_{max}.

$$D_i = D_{max}\left(1 - \frac{P_i}{\zeta + P_i}\right) \tag{4.27}$$

where ζ is an arbitrary constant controlling the curvature of the function, following Bertignac et al. (1998).

$$\sigma_i = \sqrt{2 * D_i * t} \tag{4.28}$$

Where t is the time steps we have applied the preference movement in annual cycle. This process works well for stationary preference see (FIgure **??**) The problem with this implementation is that when an individual is in a cell with low preference and no gradient signal, it will randomly jump in any direction (is this biologically sensible?). For example Tuna



(a) Preference over the spatial domain.

(b) biomass of population with the preference on the left after 30 time steps

4.7.6. Growth

Growth is the process where CABM updates both length and weight for an agent, and the same growth process can be applied over multiple time steps.

4.7.6.1. Von Bertalanffy with Basic

This process assumes that weight is an allometric function of length (Equation 4.32) and that length is an incremental function of time that follows the Von Bertalanffy growth formula. Currently it is also the default growth function used to initialise the equilibrium partition structure. The length from going from one year to the next (t represents a year) follows,

$$\Delta L_{t,i} = p_t((L_{i,\infty} - L_i)(1 - e^{-k_i})) \tag{4.29}$$

$$L_{t+1,i} = L_{t,i} + \Delta L_{t,i} \tag{4.30}$$

If you have multiple time steps in a year and you want growth over all of them, say for example you have monthly time steps and you want to apply $\frac{1}{12}$ of that increment in each year, this can be done by adding the process to all time steps and using the subcommand `time_step_proportions`. So the growth increment within a time step (k) multiplies the annual growth increment by a user defined proportion.

$$L_{t,k+1,i} = L_{t,k,i} + \Delta L_{t,i} P_k \tag{4.31}$$

Where i indexes each agents own length and growth parameters. If the basic length weight formulation is used, then an agents weight follows the following formula.

$$\bar{w}_i = a_i L_i^{b_i} \tag{4.32}$$

CABM also allow users to seed variability into agent specific growth, two parameters that are allow to vary based on an underlying **lognormal** distribution are $L_{i,\infty}$ and k_i.

$$E[L_{i,\infty}] = \text{configuration value}$$

the same goes with k_i and the variability is defined by the *CV* note that the same cv gets used for randomly drawing both parameters. Aswell as individual variability you can also seed spatial variability in these two parameters. This is done by specifying a layer in the subcommand `k_layer_label` or `linf_layer_label`. Then the expectation of the growth parameter assigned to an individual in cell j,k is

$$E[L_{i,\infty,j,k}] = \text{configuration value} \times \texttt{linf\_layer\_label}_{j,k}$$

So obviously if the layer pointed to by `linf_layer_label` are all = 1 then the expectation is the same as in the configuration file.

When an agent is created (either in initialisation or through a recruitment process) or moves cell, it gets assigned new growth parameters that relate to that cell. This allows for spatial growth, one could hypothesis that environment may explain growth and so different environments over cells will cause different rates of growth. For the growth process you must specify either a spatial layer for mean values of each growth parameter or a single value (if growth doesn't change through space), a distribution and Coefficient of variation (CV). This randomly generates an agent a parameter from that distribution, an example of the syntax in a four area model,

```
@process von_bert
type growth_von_bertalanffy_with_basic_weight
#linf_layer_label L_inf_layer
linf 101.8
#k_layer_label k_layer
```

```
k 0.161
a 2.0e-6
b 3.288
distribution lognormal
cv 0.15
time_step_proportions 1 ## if growth occurs in multiple
#time step how much growth in each time step
update_growth_parameters false ## if a individual moves area
#do we need to update its growth parameters.
```

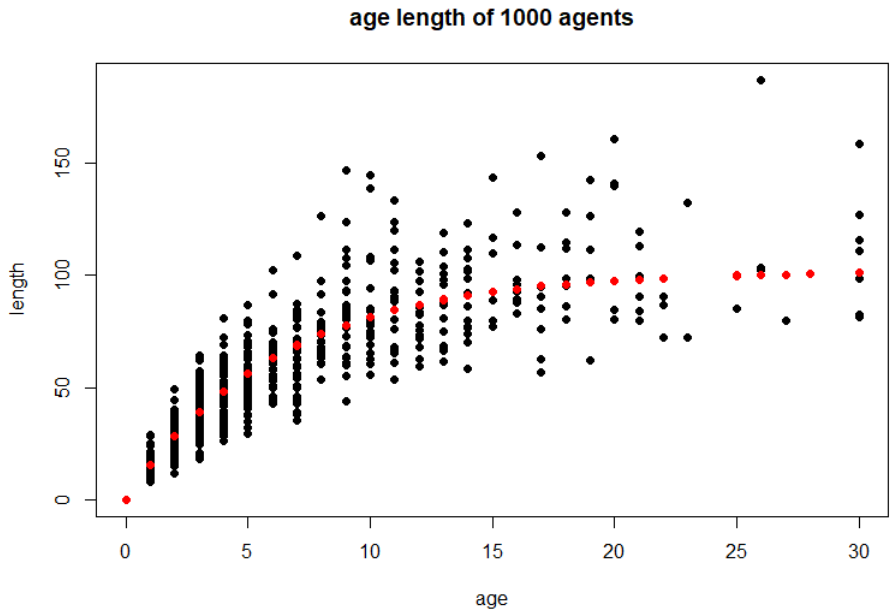


Figure 4.9: An example of the age length relationship from 1000 randomly sampled agents (black dots), red dots signify the assumed deterministic age length relationship.

4.7.6.2. Schnute with Basic

Following (Schnute, 1981) this applies the generalised Schnute growth increment model following Equation 4.33 The distribution relates to the parameters α_i and β_i for each agent. τ_1 and τ_2 are the reference ages with mean length at those reference ages defined as y_1 and y_2 respectively.

$$L_{t+1,i} = \left(L_{t,i}^{\beta_i} e^{-\alpha_i p_t} + \frac{\left(y_2^{\beta_i} - y_1^{\beta_i} e^{-\alpha_i(\tau_2-\tau_1)} \right)\left(1 - e^{-\alpha_i p_t}\right)}{\left(1 - e^{-\alpha_i(\tau_2-\tau_1)}\right)} \right)^{1/\beta_i} \tag{4.33}$$

```
# Growth
@process Growth
type growth_schnute_with_basic_weight
y1 24.5
y2 104.8
tau1 1
tau2 20
alpha 0.131
beta 1.70
```

```
t0 24.5
a 2.0e-9
b 3.288
cv 0.15
update_growth_parameters false ## if an agent moves we don't update growth info
distribution lognormal
time_step_proportions 1.0
```

4.7.7. Tagging

Tagging is a process where by agents get tagged and is a tag release event. This is a simple process where a user defines a selectivity that describes the probability of being captured for tagging, and also a number of tags that are released in each spatial cell. I have usually applied tagging before a mortality process which is where tag recaptures occur (Section 4.7.4). Also you will need to specify mortality processes to scan for tagged fish if you want any information on recapture probabilities. This will be printed in the process information.

```
@process Tagging_event
type tagging
years 1990
selectivities Sel_LL
tag_release_layer tag_1990
handling_mortality 0.12

@layer tag_1990
type integer
table layer
12314
9854
4686
end_table
```

The algorithm for this process follows

1. randomly select an agent with replacement if it is not already tagged ($U() \leq S_a$)
2. assign tag, and meta information, when it was tagged, where it was tagged etc.
3. if the agent represents multiple individuals strip it out and assign its scalar $s_i = 1$
4. See if it died from handling mortality if ($U() \leq H$) where H is the subcommand `handling_mortality`

4.8. Derived Quantities

Derived quantities surround a mortality block and so the value of the derived quantity is calculated twice, once before the mortality block (Pre-Execute), and once after (Execute). The final value is an interpolation based on how much mortality the user wants to take into account when calculating derived quantities. If users set the `proportion_through_mortality_block` parameter equal to one or zero then only one calculation is taken, and can reduce the run time of some models, so it is worth exploring.

$$B_p = (1-p)B + pB' \tag{4.34}$$

where B_p is the interpolated derived quantity after p proportion through the mortality block, B is the quantity at the beginning of the mortality block and B' is the derived quantity at the end of the mortality block.

Derived quantities can be used either to report the state of a particular components of the population, or used in populations. For example in the Beverton Holt recruitment process (Section 4.7.3.2) users must define a subcommand `ssb` which is a derived quantity. So this can be a useful tool to incorporate density dependence based processes.

Biomass

Users must specify cells in which to calculate the derived quantity over using the subcommand `biomass_layer_label`

$$B = \sum_i \text{if } (U() \leq S_a) \text{I } \bar{w}_i \times s_i \tag{4.35}$$

where for an age based selectivity where S_a is the probability of being included in this derived quantity, if the summary is age based, I is a 0 or 1 depending on the condition of the if statement, $\bar{w}_i$ is the weight of the agent and s_i is the scalar of the agent i.e. how many individuals that agent represents.

```
@derived_quantity Total_biomass_in_Summer
type biomass
biomass_layer_label Base
selectivity All
time_step Summer
proportion_through_mortality_block 0.0

@selectivity All
type constant
c 1
```

Mature Biomass

$$B = \sum_i if(mature = 1) \text{ I } \bar{w}_i \times s_i \tag{4.36}$$

where s_i is the scalar of the agent i.e. how many individuals that agent represents.

```
@derived_quantity SSB
type mature_biomass
biomass_layer_label SSB_layer
time_step time_step_one
proportion_through_mortality_block 0.0
```

Abundance

$$B = \sum_i \text{if } (U() \leq S_a) \text{I } s_i \tag{4.37}$$

where for an age based selectivity where S_a is the probability of being included in this derived quantity, if the summary is age based, I is a 0 or 1 depending on the condition of the if statement,s_i is the scalar of the agent i.e. how many individuals that agent represents.

```
@derived_quantity Total_abundance_in_Summer
type abundance
layer_label Base
selectivity All
time_step Summer
proportion_through_mortality_block 0.0

@selectivity All
type constant
c 1
```

4.9. Selectivities

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

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

The function values for some choices of parameters, for some selectivities, can result in a computer numeric overflow error (i.e., the number calculated from parameter values is either too large or too small to be represented in computer memory). CABM implements range checks on some parameters to test for a possible numeric overflow error before attempting to calculate function values. For example, the logistic selectivity is implemented such that if $(a_{50} - x)/a_{to95} > 5$ then the value of the selectivity at $x = 0$, i.e., for $a_{50} = 5$, $a_{to95} = 0.1$, then the value of the selectivity at $x = 1$, without range checking would be 7.1×10^{-52}. With range checking, that value is 0 (as $(a_{50} - x)/a_{to95} = 40 > 5$).

The available selectivities are;

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

The available selectivities are described below.

4.9.1. constant

$$f(x) = C \tag{4.38}$$

The constant selectivity has the parameter C.

4.9.2. knife_edge

$$f(x) = \begin{cases} 0, & \text{if } x < E \\ \alpha, & \text{if } x \geq E \end{cases} \tag{4.39}$$

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

4.9.3. all_values

$$f(x) = V_x \tag{4.40}$$

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

4.9.4. all_values_bounded

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ V_x, & \text{if } L \leq x \leq H \\ V_H, & \text{if } x > H \end{cases} \tag{4.41}$$

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

4.9.5. increasing

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ f(x-1) + \pi_x(\alpha - f(x-1)), & \text{if } L \leq x \leq H \\ f(\alpha), & \text{if } x \geq H \end{cases} \tag{4.42}$$

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

4.9.6. `logistic`

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

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

4.9.7. `inverse_logistic`

$$f(x) = \alpha - \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.44}$$

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

4.9.8. `logistic_producing`

$$f(x) = \begin{cases} 0, & \text{if } x < L \\ \lambda(L), & \text{if } x = L \\ (\lambda(x) - \lambda(x-1)) \,/\, (1 - \lambda(x-1)), & \text{if } L < x < H \\ 1, & \text{if } x \geq H \end{cases} \tag{4.45}$$

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

4.9.9. `double_normal`

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \tag{4.46}$$

The double-normal selectivity has parameters μ, σ_L, and σ_R. α is a scaling parameter, with default value of $\alpha = 1$. It has values α at $x = \mu$

4.9.10. `double_exponential`

$$f(x) = \begin{cases} \alpha y_0 (y_1/y_0)^{(x-x_0)/(x_1-x_0)}, & \text{if } x \leq x_0 \\ \alpha y_0 (y_2/y_0)^{(x-x_0)/(x_2-x_0)}, & \text{if } x > x_0 \end{cases} \tag{4.47}$$

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

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

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

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

4.10. Preference Functions

Preference functions are a class of equations that describe the preference for of agents to a covariate used to describe movement in the preference based movement process (Section 4.7.5.2). Currently there are only three options

1. Double normal
2. Logistic
3. Normal

These are the same equations as in the selectivities (Section 4.9), but that instead of specifying the vulnerability for age or length they describe the preference for a user defined covariate or explanatory variable so the units are completely user defined.

4.10.1. `double_normal`

$$f(x) = \begin{cases} \alpha 2^{-[(x-\mu)/\sigma_L]^2}, & \text{if } x \leq \mu \\ \alpha 2^{-[(x-\mu)/\sigma_R]^2}, & \text{if } x \geq \mu \end{cases} \tag{4.48}$$

The double-normal preference function has parameters μ, σ_L, and σ_R. α is a scaling parameter, with default value of $\alpha = 1$. It has values α at $x = \mu$.

4.10.2. `logistic`

$$f(x) = \alpha/[1 + 19^{(a_{50}-x)/a_{to95}}] \tag{4.49}$$

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

4.10.3. `normal`

$$f(x) = \alpha 2^{-[(x-\mu)/\sigma]^2} \tag{4.50}$$

The normal preference function has parameters σ, and μ. α is a scaling parameter, with default value of $\alpha = 1$. It has values α.

4.11. Time Varying Parameters

CABM has the functionality to vary some parameters annually between the start and final year of a model run. This can be for blocks of years or specific years. For years that are not specified the parameter will default to the input. Method types for time varying a parameter are; `constant`, `random_walk`, `exogenous`, `linear`, `annual_shift`, `random_draw`. This allows users to let a parameter be known in a year, be the result of a deterministic equation, or stochastic.

When allowing removals to have annual varying, selectivities and other more realistic model components, simulated observations more closely model real data and associated conclusions become more useful. Another driver for implementing time-varying parameters was allowing mean or location parameters of selectivities to change between years based on an explanatory variable. An example of this is in the New Zealand Hoki fishery where we allow the μ and a_{50} parameters to shift depending on when the fishing season occurs. Descriptive analysis showed that when fishing was earlier relative to other years smaller fish were caught and vice versa.

4.11.1. `constant`

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

```
@time_varying m_time_var
type constant
parameter process[natural_mortality].m
years 1975:1988
values 0.123
```

4.11.2. `random_walk`

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

```
@time_varying m_time_var
type random_walk
parameter process[natural_mortality].m
distribution normal
mean 0.123
sigma 0.05
```

A constraint whilst using this functionality is that a parameter cannot be less than 0.0, if it is CABM sets it equal to 0.01.

4.11.3. `annual_shift`

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

$$\bar{\theta}_y = \frac{\sum_y^Y \theta_y}{Y} \tag{4.51}$$

$$\theta'_y = a\bar{\theta}_y + b\bar{\theta}_y^2 + c\bar{\theta}_y^3 \tag{4.52}$$

4.11.4. exogenous

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

$$\delta_y = a(E_y - \bar{E}) \tag{4.53}$$

$$\theta'_y = \theta_y + \delta_y \tag{4.54}$$

where δ_y is the shift or deviation in parameter θ_y in year y to generate the new parameter value in year y (θ'_y). a is an estimable shift parameter, E is the exogenous variable and E_y is the value of this variable in year y. For more information readers can see Francis et al. (2003).

4.12. Tips setting up configuration files

CABM can take a while if you have a complex spatial model and complex life history. So there are some tips and tricks that I have come found can be useful when setting up configuration files.

Debugging a model

When initially setting up a model set initialisation years low and number of agents low, because there is a lot going on in this model it is best to get up and run (which is a good feeling). Also I would advice setting up the population and report components, only once you are happy with that that I would suggest bringing in Observations. Advice I always struggle to keep to is, start very basic and build complexity, otherwise say good bye to time and hello to frustration. Once you are happy with your model then I would suggest increase initialisation years and number of agents.

The initialisation Phase

The first thing you wont to nail down before getting gun hoe on the configuration and complex dynamics you want to get a model that reaches an appropriate initialisation state *quickly!*. So my suggestion is to not worry about fishing in the first instance, set the `start_year` and `final_year` one year apart and play with the initialisation phase commands to see how to efficiently reach a desired initial state. The parameters I am talking about are the subcommands `years` and `layer_label`. The `years` parameter sets how many annual cycles to run through to set your initialisation state, you want this as small as possible. So my advice is run the model with a range of `years` say 5, 10, 20, 30, 40, 50 and 100 to find out how sensitive your initial state is to this. You find that you have to make a compromise between speed and accuracy, unfortunately life is full of such compromises. To see some example **R** code on how I compare initialisation see Section 14.

Along with the `years` parameter you can also set the spatial distribution of the initial seeding of agents through the `layer_label`, if you have a spatial model with movement this would be an interesting one to play with. So pretty much my advice is tweak these initialisation parameters as much as you can before you move onto fishing and generate observations, it will be well worth your while if you treasure time.

The number of agents in the system

This one should be obvious and we suggest that you play with the initialisation parameter `number_of_agents` to see how sensitive model outputs are to this parameter. The less agents in the system the faster your model will run, however the less agents the more coarse you model becomes because then an entity all of a sudden represents 1000 if not 10000 entities, so once again we return to a compromise.

The number of threads available

Disabled was not implemented correctly the first time, still to many shared resources..

more ways than one to skin a cat

Hopefully throughout the process and derived quantities I have discussed alternative ways to set up a similar model. Given there generality of the program there are more than one way to approximate dynamics and some will be more computationally quicker than others. So have a play around but try and keep the number of times we iterate over the state as small as possible, as you will get significant time saving. It is also possible to make your own specific process that does a range of processes at one time, thus saving a lot of time. This can be easily done by looking over the current processes and taking the functionality that has been tested and copy them in to a new file.

5. The estimation section

5.1. The numerical differences minimiser

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

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

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

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

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

6. The observation section

This section describes the observations that CABM can generate during run time. CABM calculates expectations of all the agents that are user defined, and then adds observation error via simulating through a distribution with a user defined observation error value.

6.1. Observations

6.1.1. Process Removals By Age

This observation class aggregates age frequency over a user defined spatial area from a `mortality_event_biomass` and `mortality_baranov` process. This class can add ageing error onto the expectation, to account for ageing error which is a source of uncertainty in ageing fish. An example of how you would set this observation up and show some of the flexibilities in the spatial resolution see the syntax below for a 6 area model.

```
@process fishing
type mortality_event_biomass
years 2000
catch_layers catch_2000
selectivity fishing_selectivity

@layer catch_2000
type numeric
table_layer
600 500 400
1034 601 200
903 450 100
end_table

@layer cells
type categorical
table_layer
r1-c1 r1-c2 r1-c3
r2-c1 r2-c2 r2-c3
r3-c1 r3-c2 r3-c3
end_table

@observation fishery_age
type mortality_event_composition
layer_of_stratum_definitions cells
strata_to_include r1-c1
simulation_likelihood multinomial
process_label fishing
years 2000
plus_group true
table error_values
year r1-c1
2000 1000
end_table
```

The above syntax asks for an age frequency in just the top left cell (`r1-c1`). You could easily summarise the age frequency over the whole spatial domain by changing the categorical layer as shown below

```
@layer cells
type categorical
table_layer
single_cell single_cell single_cell
single_cell single_cell single_cell
single_cell single_cell single_cell
end_table

@observation fishery_age
type mortality_event_composition
layer_of_stratum_definitions cells
strata_to_include single_cell
simulation_likelihood multinomial
process_label fishing
years 2000
plus_group true
table error_values
year single_cell
2000 1000
end_table
```

Hopefully the above example illustrates how much control the user has in specifying what information they want to extract.

6.1.2. Biomass

This observation summarises the biomass or abundance over a selected number of agents in a time step over the mortality block (Section 4.6.2), for user defined spatial areas. CABM does an interpolation similar to the derived quantities if the user wishes to extract a biomass observation that represents a proportion of mortality being taken.

For abundance observations, there is a command `abundance` in the biomass type observation that can be set to true.

```
@observation survey_index
type biomass
years 1990:2013
time_step Summer
abundance false
catchability 0.342
proportion_through_mortality_block 1.0 ## take snapshot at the end of timestep
simulation_likelihood lognormal
error_value 0.2 * 24
selectivities Sel_survey
cell_layer cells
cells r1-c1 r2-c1 r3-c1
```

6.1.3. Proportions at age

This observation summarises the proportions at age for selected number of agents in a time step over the mortality block (Section 4.6.2)

```
@observation survey_age_comp
```

```
type proportions_at_age
simulation_likelihood multinomial
years 1990:1995
min_age 0
max_age 20
plus_group true
ageing_error none
table error_values
1990 300
1991 300
1992 300
1993 300
1994 300
1995 300
end_table
cell_layer cells
cells r1-c1 r2-c1 r3-c1
```

6.1.4. Age frequency from scaled length frequency

This observation class, generates an age frequency by getting all agents removed by an F method in user defined stratum, and returns a sub-sample of age and lengths that is used to generate an age length key. The user can ask for ageing-error to be applied to when generating an age length key, using the `@ageing_error` block (see Section 6.2). Calculations currently follow the following algorithm for each spatial stratum,

1. randomly sub sample fishery to calculate length frequency
2. within the sampled fish for length frequency, sub sample ages based on the `age_allocation_method` command
3. apply ageing error to the ages.

$$N_{n,a} = \sum_{l} K_{a,l,n} N_{l,n} \tag{6.1}$$

where $K_{a,l,n}$ is the age length key for stratum n, $N_{l,n}$ is the numbers at length which can be a sub sample of the fishery, this is controlled by the input table `proportion_lf_sampled`. There are three methods for allocating ages for a single age length key, these are `random`, `equal`, `proportional`. Random will uniformly sample without replacement, Equal will put equal amount of ages in each length bin that is non-zero, and proportional will distribute the number of ages, proportional to the length frequency.

$$N_{a} = \sum_{n} \sum_{s} N_{n,a,s} \tag{6.2}$$

At some point when I return to this I wont to add sex if sexual dimorphism is evident among agents, see below for more improvements that I will be adding to this class.

```
@observation scaled_age_freq
type mortality_scaled_age_frequency
years 1991
process_label summer_fishery
ageing_error none
stratum_weighting_method biomass
layer_of_stratum_definitions summery_fishery_catch_at_age_stratum
stratums_to_include Inshore Offshore
age_allocation_method proportional

table samples ## number of individuals randomly selected to calculate age-length key
year   Inshore    Offshore
1991   400        200
end_table

table proportion_lf_sampled ## proportion of catch that is sampled for Length frequency
year   Inshore    Offshore
1991   0.8        0.7
end_table
simulation_likelihood multinomial

@layer summery_fishery_catch_at_age_stratum
type categorical
table layer
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
end_table
```

6.1.5. Age frequency from scaled length frequency for Mortality Event Biomass process

This observation class, is an extension of the class Age frequency from scaled length frequency (Section 6.1.4) and is the recommended age-length method when generating mortality based age-frequencies from process of type `mortality_event_biomass`. This is mainly because the process `mortality_event_biomass` allows for multiple fisheries (gear types/vulnerabilities) and so generally we want to generate an age-frequency for a specific fishery. The algorithm is similar to that of the class is derives from with the following steps run.

For each Stratum identify all agents that are measured for Length frequency.

1. randomly sub sample cells that are within each stratum (can have many)
2. For each cell randomly select all available agents for measuring length frequency (denoted by table `proportion_lf_sampled`)

For all agents measured for length within the stratum (N_n) sub-sample agents available for otolith reading with a length based probability according to the subcommand `age_allocation_method`. Where the probabilities are defined for an agent of length l as,

1. `random` $\frac{1}{N_n}$
2. `proportional` $\frac{N_{l,n}}{N_n}$

3. `equal` $\frac{1}{n_l}$

where $N_{l,n}$ is the numbers measured for length in length bin l in stratum n and n_l is the number of length bins that are non zero. When sub-sampling for age occurs this sampling is **without** replacement. The algorithm will do it's best to allocate ages according to these probabilities, if after so many attempts it can't fully allocate the number of otoliths according to these probabilities. It will abandon random methods and try systematically find fish to complete the distribution. This will occur when is a low probability that by its very definition will take a long time to randomly generate a sample of. Once we have randomly sub-sampled ages we calculate an Age Length Key (ALK) (represented as proportions) and calculate final ages following. Ageing error is applied in the age-length key, so if an agent is of age 3 but misclassified as age 4 then it is put in the ALK as age 4.

$$N_{n,a} = \sum_{l} K_{a,l,n} N_{l,n} \tag{6.3}$$

$$N_a = \sum_{n} \sum_{s} N_{n,a,s} \tag{6.4}$$

```
@observation scaled_age_freq
type mortality_event_biomass_scaled_age_frequency
years 1991
process_label summer_fishery
fishery_label fisheryTrwl
ageing_error none
stratum_weighting_method biomass
layer_of_stratum_definitions summery_fishery_catch_at_age_stratum
stratums_to_include Inshore Offshore
age_allocation_method proportional

table samples ## number of individuals randomly selected to calculate age-length key
year   Inshore    Offshore
1991   400        200
end_table

table proportion_lf_sampled ## proportion of catch that is sampled for Length frequency
year   Inshore    Offshore
1991   0.8        0.7
end_table
simulation_likelihood multinomial

@layer summery_fishery_catch_at_age_stratum
type categorical
table layer
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
Inshore Inshore Offshore Offshore Offshore
end_table
```

Cool things I will plan to do with this observation class

This will be a great class for investigating catch at age inputs, I will have to return to this as this is not yet in my scope of project but things you would need to add before this observation is useful or representative of applied protocol,

- ~~Add sex in these equations.~~
- ~~currently we only get spatial stratum based information, but somehow structuring fishery data to represent tows and fleets mights be quite handy in this type of investigation.~~
- different methods for weighting, to generate scaled length frequency you would want to add the options to weight stratum by biomass, area, numbers or proportion
- Bootstrap estimates, we could boot strap ALK and Scaled length frequencies to get CV for each age bin in the frequency, and calculate an effective sample size.

where, u is a random number generated from uniform random variable on $[0,1]$. This algorithm is shown to generate unbiased estimates of the target distribution (Figure 6.2).

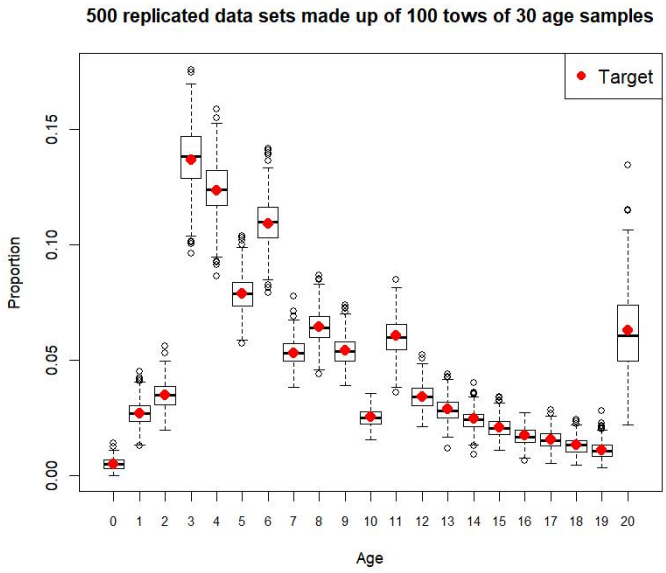
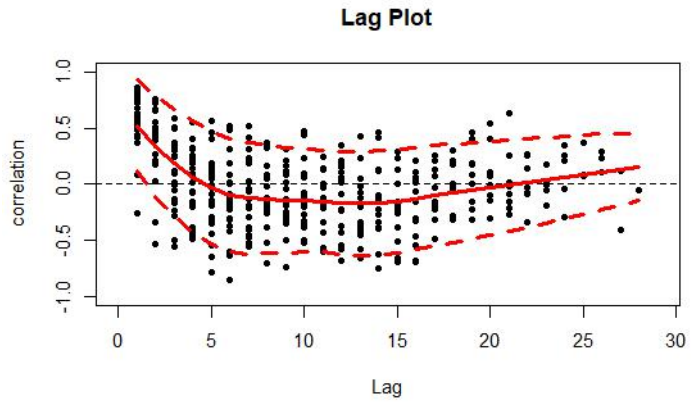
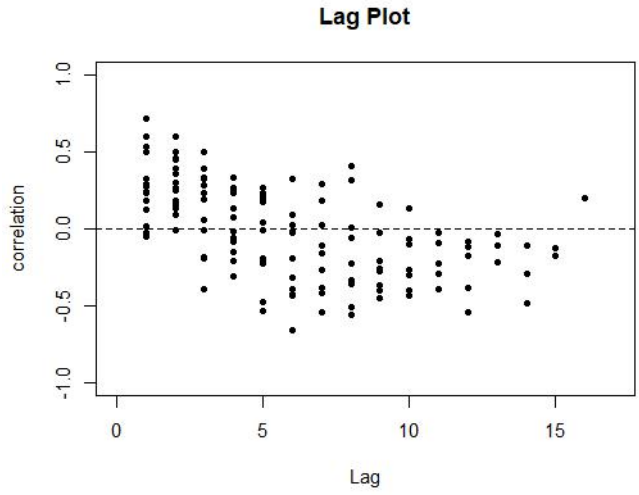


Figure 6.1: Boxplots of simulated age frequency using the algorithm described in the methods, with $n_a = 30$ (number of otoliths per cluster), $n_c = 100$ (number of clusters) and $\sigma_c = 2.5$, showing that on average they generate unbiased proportions at age.



(a)



(b)

Figure 6.2: Top panel (a) are, Pearson residuals from a stock assessment assuming a multinomial distribution, with data simulated from the IBM with this cluster algorithm. The bottom panel (b)are Pearson residuals, from the HAK 3&4 stock assessment from the Chatham Rise Age composition data set.

6.1.6. Age frequency from Mortality Event Biomass process with clustering

In this observation, you specify the number of clusters which can be thought of as hauls you want to sample from a fishing event (specific fishery and time), as well as some attributes of clusters i.e. `average_cluster_weight` in tonnes average weight of haul to sample, the cv `cluster_cv` which describes the variance around the mean weight of hauls to sample from (distribution assumed is log normal). Within each cluster you can specify, how many otoliths will be sampled from each haul, and how correlated the ages will be within a haul, this is defined by the `cluster_sigma`. As this parameter goes to zero the hauls are highly homogeneous, compared to a large value, where the ages become uncorrelated. You can ask for the observation to be normalised as proportions (`composition_type proportions`) or as numbers.

The other neat thing regarding this observation class, is you can weight each haul,

The algorithm

- Iterate over each stratum, calculating weight in each stratum, and associated cells with their weights
- for each stratum

  iterate over all cells within each strata

  with in each cell

  randomly select a cluster/haul, number of hauls per cell are weighted by how much biomass was in the cell compared to all cells within strata

  Randomly select a cluster size (weight based quantity)

  Calculate that to abundance using a simple mean weight per agent in the cell.

  Randomly select on agent in available in the current cell, this is the starting agent, and all other agents in the cluster are conditional on this first agent (thus creating a school effect)

  We implement a Metropolis Hastings algorithm (steps 6.1.6) to select the number agents that the user desires for this cluster.

  Can apply ageing error at this point

  weight each aged fish's contribution by the weight of the cluster e.g. an aged fish from a cluster of 100 tonnes will have more 'weight' than an aged fish from a cluster of 1 tonne.

1. Randomly select an individual x_0 from the population (all individuals have equal probability of capture)
2. generate a proposal individual from $x_i' \sim \mathcal{N}(x_{i-1}, \sigma_c^2)$
3. calculate the acceptance ratio $\alpha = \frac{\pi(x_i')}{\pi(x_{i-1})}$
4. accept if $u \leq \alpha\ x_i = x_i'$ or reject if $u > \alpha\ x_i = x_{i-1}$
5. repeat steps 2-4 until an adequate sample size is reached denoted by n_a.
6. repeat steps 1-5 to generate multiple clusters n_c

6.1.7. Tag-recaptures by length

6.1.8. Tag-recaptures by age

6.2. Ageing error

CABM can apply ageing error to an expected age frequency generated by the model. The ageing error is applied as a misclassification matrix, which has the effect of 'smearing' the expected age frequencies. This is mimicking the error involved in identifying the age of individuals. For example fish species are aged by reading the ear bones (otoliths) which can be quite difficult depending on the species. These are used in generating age based observations.

Ageing error is optional, and if it is used, it may be omitted for any individual time series. Different ageing error models may be applied for different observation commands. See Section **??** for reporting the misclassification matrix at the end of model run.

The ageing error models implemented are,

1. None: The default model is to apply no ageing error.

2. Off by one: Proportion p_1 of individuals of each age a are misclassified as age $a-1$ and proportion p_2 are misclassified as age $a+1$. Individuals of age $a < k$ are not misclassified. If there is no plus group in the population model, then proportion p_2 of the oldest age class will 'fall off the edge' and disappear.

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

Note that the expected and simulated observations reported by CABM for observations with ageing error will have had the ageing error applied.

7. The report section

The report section specifies the printouts and other outputs from the model. CABM does **not** produce any output unless requested by a valid `@report` block. This important to note, as model runs can take a few minutes to run it will be in vain if there are no reports.

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

Also remember because this is a terminal run program to write pipe out the output to file like `ibm -r > run.out` else it would print to the terminal and be fairly useless. Once you have CABM output in a text file like the above example `run.out`, go to Section 14 for how to use the R-library for reading in and plotting information in the **R** environment.

7.0.1. Initialisation Partition

A very useful report for looking at the initialisation partition. This report prints too instances of the partition. The first is right after we do an exponential approximation in each cell, and the second is after we have run the initial annual cycle for `years` and are about to enter the actual annual cycle.

```
@report init
type initialisation_partition
```

7.0.2. Age Frequency

There are two reports for printing the age frequency, you can print it for the all cells the total age frequency at the end of a time step using the `type world_age_frequency`. Or you could choice to print the age frequency for each cell using the `type age_frequency_by_cell`.

```
# print age frequency by cell
@report age_freq
type age_frequency_by_cell
#years 1990:2018 ## defaults to model years
time_step Annual

# print age frequency for all cells
@report world_age_freq
type world_age_frequency
#years 1990:2018 ## defaults to model years
time_step Annual
```

7.0.3. Observation

This prints a summary of an observation and simulated values, for each year and cell.

```
@observation fishery_age
```

```
type process_removals_by_age
simulation_likelihood multinomial
process_label fishing
years 2000
min_age 0
max_age 28
#plus_group true
ageing_error None
table error_values
2000 10000
end_table
cell_layer cells
cells r1-c1

@report fisher_age_freq
type observation
observation fishery_age
```

7.0.4. Model Attributes

This pretty much just prints the global scalar used to convert numbers and biomass from agent space to population level.

```
@report model_attributes
type model_attributes
```

7.0.5. Derived Quantities

This report will print all the derived quantities for all years in the model.

```
@report derived_quants
type derived_quantity
```

7.0.6. Process

This report will print all parameters that were set for a specific process and any auxiliary information that you want from the specific process. Many of the processes will print extra information such as some of the mortality processes that are responsible for tag recaptures.

```
@report M_report
type process
process natural_mort
```

7.0.7. Numeric Layer

This report will print out the model derived numeric layers such Abundance and biomass. Useful for showing the spatial distribution of all the agents over time.

```
@report biomass_by_cell
type numeric_layer
layer_label biomass_by_cell
years 1990:2018
time_step Summer
```

7.0.8. Time varying

This report will print out all the time varying parameters and the values assumed in each year

```
@report time_varying_parameters
type time)varying
```

7.0.9. Summarise Agents

This report will randomly print all the attributes from a user defined number of agents that can be viewed at the end of each time step. Useful looking at length weight by area, tagged fish etc.

```
@report agents
type summarise_agents
years 1990:2018
time_step Summer
number_of_individuals 1000
```

8. The Management Strategy Evaluation (MSE)

8.1. Introduction

A key run mode of the CABM is the MSE `ibm -m` run mode. This run mode creates an interface with an **R** console. At the end of each `assessment_year` the CABM pauses and waits for the **R** scripts run and return a catch to apply for the years up until the next `assessment_years`. Currently the only mortality process available for this run mode is `mortality_event_hybrid` (Section 4.7.4.6), which requires users to supply fishing mortality (`mortality_baranov`) by fisheries between `start_year` and `assessment_year` and then applies a catch based mortality (`mortality_event_biomass`) over the remaining `assessment_year`.

8.2. Configuration

This run mode expects a specific directory structure and set of files available. Assuming the CABM configuration models are in the directory called `ibm` the program expects there to be a directory up one level titled `R`. If you look at the source code `Model.cpp` line: 444 `source(file.path('..','R','SetUpR.R'))`. In the `R` directory there must be the following **R** files which are sourced in the CABM program these files are case sensitive.

- `SetUpR.R` executed once, users should load R libraries and set up global variables that you don't want to keep creating
- `ResetHCRVals.R` executed between each `-i`
- `RunHCR.R` executed at the end of each year in the `assessment_years`

Specifying the `@model` block as follows.

```
@model
start_year 1975 #
final_year 2036
assessment_years 2016 2020 2024 2028 2032 2036
```

This configuration will run the basic (`ibm -r`) run mode until the year 2016. After each year in the `assessment_years` subcommand the Rscript `R\RunHCR.R` is executed where the CABM expects catches to be returned for the years up until the next year in `assessment_years` i.e. in 2016 it expects catches for 2017→2020.

The fishing mortality process `mortality_event_hybrid` only needs the table `table f_table` up to the first value in the `assessment_years`. After the first assessmnet year the catches will come from **R**. However you will still need to have to specify other processes such as recruitment for all final years, the same as `@observation` classes.

- Set up internal **R** console and run the **R** script `R\SetUpR.R`
- Run from `start_year` to the first year in the `assessment_years`.
- At the end of each year in the `assessment_years` the CABM will
    1. simulate data based on both users specifying the correct `@observation` and `@report` configurations

2. execute `R\RunHCR.R`
3. Run the CABM forward upto and including the next year in the `assessment_years`

This run mode can be run in-conjunction with multiple inputs `-i` parameters to account for parameter uncertainty. There is an example model in the `Examples\MSE` for a complete model run.

Notes with regard to the R interface. The key R script is `R\RunHCR.R`. The structure of the return object that the CABM expects is a nested list with the top level being year, second level being fishing label (which needs to match the table fishing label). Note don't change the working directory in **R** else it will mess the path for the C++ program. To debug **R** script you can use the `cat` command, however this will print out into the report configuration and mess up the R-library when reading in the output.

```
## exert of R\RunHCR.R
future_catch = list() # the list that will be returned
## Here is code that takes simulated data and creates an
## estimate of stock status, with future catches.
## the R knows variables passed by the C++ interface
## 'current_year', 'next_ass_year'
fishing_vector = 2000
names(fishing_vector) = "Fishing" # name the vector based on fisheries
for(year_ndx in (current_year + 1):next_ass_year)
Fishing_ls[[as.character(year_ndx)]] = fishing_vector
## return the list which the C++ will pick up.
Fishing_ls
```

9. Population command and subcommand syntax

9.1. Model structure

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

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

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

`assessment_years` Years to conduct HCR in, Is conducted at the end of the year, so HCR should supply catch rules for the following year
Type: non-negative integer vector
Default: true

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

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

`age_plus` Define the oldest age or extra length midpoint (plus group size) as a plus group
Type: boolean
Default: false
Value: true, false

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

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

length_bins
Type: non-negative integer vector
Default: false

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

base_layer_label Label for the base layer
Type: std::string
Default: No Default

latitude_bounds Latitude bounds for the spatial domain, should include lower and upper bound, so there should be rows + 1 values
Type: constant vector
Default: true

longitude_bounds Longitude bounds for the spatial domain, should include lower and upper bound, so there should be columns + 1 values
Type: constant vector
Default: true

nrows number of rows in spatial domain
Type: non-negative integer
Default: No Default
Lower Bound: 1 (inclusive)

ncols number of columns in spatial domain
Type: non-negative integer
Default: No Default
Lower Bound: 1 (inclusive)

sexed Is sex an attribute of then agents?
Type: boolean
Default: false

growth_process_label Label for the growth process in the annual cycle
Type: std::string
Default: No Default

natural_mortality_process_label Label for the natural mortality process in the annual cycle
Type: std::string
Default: No Default

mortality_process_for_mse Label for the mortality process that is used in MSE methods

Type: std::string
Default: ""

9.2. Initialisation

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

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

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

9.2.1. @initialisation_phase[label].type=iterative

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

initial_number_of_agents The number of agents to initially seed in the partition
Type: non-negative integer
Default: No Default

layer_label The label of a layer that you want to seed a distribution by.
Type: std::string
Default: ""

recruitment_layer_label The label of a layer has a recruitment process label in each cell to see how to set scalars
Type: std::string
Default: ""

initialisation_mortality_rate The instaneous mortality rate to use to approximate a crude initial age-structure
Type: float
Default: 0.2

9.3. Time-steps

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

label The label of the timestep

Type: std::string
Default: No Default

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

9.4. Processes

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

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

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

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

9.4.2. `@process[label].type=growth_schnute_with_basic`

`distribution` the distribution to allocate the parameters to the agents
Type: std::string
Default: normal
Allowed Values: normal, lognormal

`update_growth_parameters` If an agent/individual moves do you want it to take on the growth parameters of the new spatial cell
Type: boolean
Default: No Default

`cv` The cv of the distribution
Type: constant
Default: No Default

`alpha_layer_label` Label for the numeric layer that describes mean L inf by area
Type: string vector
Default: true

`beta_layer_label` Label for the numeric layer that describes mean k by area
Type: string vector
Default: true

`t0_layer_label` Label for the numeric layer that describes mean t0 by area
Type: string vector
Default: true

`a_layer_label` Label for the numeric layer that describes mean a in the weight calcualtion through space
Type: string vector
Default: true

`b_layer_label` Label for the numeric layer that describes mean b in the weight calcualtion through space
Type: string vector
Default: true

`t0` The value for t0 default = 0
Type: constant vector
Default: true

`alpha` alpha value for schnute growth curve
Type: Addressable vector
Default: true

`beta` beta value for schnute growth curve
Type: Addressable vector
Default: true

`a` alpha value for weight at length function
Type: constant vector
Default: true

`b` beta value for weight at length function
Type: constant vector
Default: true

`tau1` reference age for y1
Type: Addressable vector
Default: No Default

`tau2` reference age for y2
Type: Addressable vector
Default: No Default

`y1` mean size at reference ages tau1
Type: constant vector
Default: No Default

y2 mean size at reference ages tau2
Type: constant vector
Default: No Default

alpha
Type: Addressable vector
Default: No Default

beta
Type: Addressable vector
Default: No Default

tau1
Type: Addressable vector
Default: No Default

tau2
Type: Addressable vector
Default: No Default

9.4.3. @process[label].type=growth_von_bertalanffy_with_basic

distribution the distribution to allocate the parameters to the agents
Type: std::string
Default: normal
Allowed Values: normal, lognormal

update_growth_parameters If an agent/individual moves do you want it to take on the growth parameters of the new spatial cell
Type: boolean
Default: No Default

cv The cv of the distribution
Type: constant
Default: No Default

linf_layer_label Label for the numeric layer that describes mean L inf by area
Type: string vector
Default: true

k_layer_label Label for the numeric layer that describes mean k by area
Type: string vector
Default: true

t0_layer_label Label for the numeric layer that describes mean t0 by area

Type: string vector
Default: true

`t0` The value for t0 default = 0
Type: constant vector
Default: true

`a_layer_label` Label for the numeric layer that describes mean a in the weight calcualtion through space
Type: string vector
Default: true

`b_layer_label` Label for the numeric layer that describes mean b in the weight calcualtion through space
Type: string vector
Default: true

`linf` Value of mean L inf multiplied by the layer value if supplied
Type: Addressable vector
Default: true

`k` Value of mean k multiplied by the layer value if supplied
Type: Addressable vector
Default: true

`a` alpha value for weight at length function
Type: constant vector
Default: true

`b` beta value for weight at length function
Type: constant vector
Default: true

`linf`
Type: Addressable vector
Default: No Default

`k`
Type: Addressable vector
Default: No Default

9.4.4. @process[label].type=maturity

9.4.5. @process[label].type=mortality_baranov

print_census_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

9.4.6. @process[label].type=mortality_baranov_simple

years years to apply the fishery process in
Type: non-negative integer vector
Default: No Default

fishing_mortality_layers Spatial layer describing F by area for each year, there is a one to one link with the year specified, so make sure the order is right
Type: string vector
Default: No Default

minimum_legel_length The minimum legal length for this fishery, any individual less than this will be returned using some discard mortality
Type: constant
Default: 0

handling_mortality if discarded due to being under the minimum legal length, what is the probability the individual will die when released
Type: constant
Default: 0.0

print_extra_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

scanning_proportion_of_catch The proportion of catch scanned in each year
Type: constant vector
Default: true

cv The cv of the distribution for the M distribution
Type: constant
Default: No Default

m_multiplier_layer_label Label for the numeric layer that describes a multiplier of M

through space
Type: std::string
Default: ""

natural_mortality_selectivity_label Selectivity label for the natural mortality process
Type: string vector
Default: No Default

m Natural mortality for the model
Type: estimable
Default: No Default

9.4.7. @process[label].type=mortality_constant_rate

distribution the distribution to allocate the parameters to the agents
Type: std::string
Default: normal
Allowed Values: normal, lognormal

cv The cv of the distribution
Type: constant
Default: No Default

m_multiplier_layer_label Label for the numeric layer that describes a multiplier of M through space
Type: std::string
Default: ""

m Natural mortality for the model
Type: Addressable
Default: No Default

update_mortality_parameters If an agent/individual moves do you want it to take on the natural mortality parameters of the new spatial cell
Type: boolean
Default: No Default

time_step_ratio Time step ratios for the mortality rates to apply in each time step. See manual for how this is applied
Type: constant vector
Default: No Default

m
Type: Addressable
Default: No Default

9.4.8. @process[label].type=mortality_cull

layer_label Label for the integer layer which we will kill all agents in
Type: std::string
Default: // TODO perhaps as a multiplier

9.4.9. @process[label].type=mortality_effort_based

selectivity Selectivity label
Type: string vector
Default: No Default

minimiser Label of the minimser to solve the problem
Type: std::string
Default: No Default

years years to apply the process
Type: non-negative integer vector
Default: No Default

catches Total catch by year
Type: constant vector
Default: No Default

effort_values A vector of effort values, one for each enabled cell of the model, these should represent the variability of effort of the fishery mimicking
Type: constant vector
Default: true

effort_layer_label A layer label that is a numeric layer label that contains effort values for each year.
Type: std::string
Default: ""

starting_value_for_lambda Total catch by year
Type: constant vector
Default: true

9.4.10. @process[label].type=mortality_effort_based_with_covar

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

`minimiser` Label of the minimser to solve the problem
Type: std::string
Default: No Default

`years` years to apply the process
Type: non-negative integer vector
Default: No Default

`catches` Total catch by year
Type: constant vector
Default: No Default

`effort_values` A vector of effort values, one for each enabled cell of the model, these should represent the variability of effort of the fishery mimicking
Type: constant vector
Default: No Default

`starting_value_for_lambda` Total catch by year
Type: constant vector
Default: true

`preference_functions` The preference functions to apply to each layer
Type: string vector
Default: No Default

`preference_layers` The preference functions to apply
Type: string vector
Default: No Default

`preference_weights` The weight to each preference when calculating mean spatial effort distribution
Type: constant vector
Default: No Default

`closure_layer` Layer label for closed areas
Type: std::string
Default: ""

9.4.11. @process[label].type=mortality_event_biomass

print_census_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

print_tag_recapture_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

9.4.12. @process[label].type=mortality_event_hybrid

print_census_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

9.4.13. @process[label].type=mortality_exploitation

print_census_info if you have process report for this process you can control the amount of information printed to the file.
Type: boolean
Default: true

9.4.14. @process[label].type=movement_box_transfer

origin_cell The origin cell associated with each spatial layer (should have a one to one relationship with specified layers), format follows row-col (1-2
Type: string vector
Default: No Default

probability_layers Spatial layers (one layer for each origin cell) describing the probability of moving from an origin cell to all other cells in the spatial domain.
Type: string vector
Default: No Default

movement_type What type of movement are you applying?
Type: std::string
Default: markovian
Allowed Values: markovian, natal_homing

selectivity_label Label for the selectivity block

Type: string vector
Default: No Default

9.4.15. @process[label].type=movement_preference

d_max The maxiumu diffusion value
Type: constant
Default: No Default
Lower Bound: 0.0 (exclusive)

time_intervals The time interval that this movement process occurs over (k) in the documentation formula's
Type: constant
Default: 1.0
Lower Bound: 0.0 (exclusive)

zeta An arbiituary parameter that controls curvature between preference and diffusion
Type: constant
Default: 1.0
Lower Bound: 0.0 (exclusive)

brownian_motion Just apply random movement, undirected movement
Type: boolean
Default: false

preference_functions The preference functions to apply
Type: string vector
Default: true

preference_layers The preference functions to apply
Type: string vector
Default: true

selectivity_label Label for the selectivity block
Type: string vector
Default: No Default

save_and_print_a_sample_of_agent_jumps Save an location after each movement application for each agent, and print 30 of them in the report (Can be very slow
Type: boolean
Default: false

9.4.16. @process[label].type=nop

9.4.17. @process[label].type=recruitment_beverton_holt

r0 R0
Type: Addressable
Default: No Default

recruitment_layer_label A label for the recruitment layer, that describes spatial distribution of recruits.
Type: std::string
Default: No Default

ssb A label for the SSB derived quantity
Type: std::string
Default: No Default

proportion_male Proportion of recruits male
Type: Addressable vector
Default: 1.0

r0
Type: Addressable
Default: No Default

proportion_male
Type: Addressable vector
Default: No Default

steepness Steepness
Type: constant
Default: 1.0

ycs_values YCS (Year-Class-Strength) Values
Type: estimable vector
Default: No Default

9.4.18. @process[label].type=recruitment_constant

r0 R0
Type: Addressable
Default: No Default

recruitment_layer_label A label for the recruitment layer, that describes spatial distribution

of recruits.
Type: std::string
Default: No Default

`ssb` A label for the SSB derived quantity
Type: std::string
Default: No Default

`proportion_male` Proportion of recruits male
Type: Addressable vector
Default: 1.0

`r0`
Type: Addressable
Default: No Default

`proportion_male`
Type: Addressable vector
Default: No Default

9.4.19. `@process[label].type=tag_shedding`

`selectivity_label` Label for the selectivity block
Type: string vector
Default: No Default

`time_step_ratio` Time step ratios for the Shedding rates to apply in each time step. See manual for how this is applied
Type: constant vector
Default: No Default

`shedding_rates` Shedding rate per Tag release event
Type: constant vector
Default: No Default

`tag_release_region` The Release region for the corresponding shedding rate, in the format 1-1 for the first row and first column, and '5-2' for the fifth row and secnd column
Type: string vector
Default: No Default

`tag_release_year` The Release Year for the corresponding shedding rate
Type: non-negative integer vector
Default: No Default

`years` Years to execute the tag shedding process

Type: non-negative integer vector
Default: No Default

`stop_tracking_tags_in_year` Stop Tracking tagged partition in this year, they will just die. But will stop containimation in observations.
Type: non-negative integer vector
Default: No Default

9.4.20. `@process[label].type=tagging`

`tag_release_layer` Spatial layer describing catch by cell for each year, there is a one to one link with the year specified, so make sure the order is right
Type: string vector
Default: true

`selectivities` selectivity used to capture agents
Type: string vector
Default: ""

`handling_mortality` What is the handling mortality assumed for tagged fish, sometimes called initial mortality
Type: constant
Default: 0

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

9.5. Time varying parameters

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

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

`type` The time-varying type
Type: std::string
Default: ""

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

`parameter` The name of the parameter to time vary

Type: std::string
Default: No Default

9.5.1. @time_varying[label].type=annual_shift

values
Type: constant vector
Default: No Default

a
Type: constant
Default: No Default

b
Type: constant
Default: No Default

c
Type: constant
Default: No Default

scaling_years
Type: non-negative integer vector
Default: true

9.5.2. @time_varying[label].type=constant

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

9.5.3. @time_varying[label].type=exogenous

a Shift parameter
Type: estimable
Default: No Default

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

9.5.4. @time_varying[label].type=linear

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

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

9.5.5. @time_varying[label].type=random_draw

`mean` Mean
Type: estimable
Default: 0

`sigma` Standard deviation
Type: estimable
Default: 1

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

`lower_bound` Lower bound
Type: constant
Default: No Default

`upper_bound` Upper bound
Type: constant
Default: No Default

9.5.6. @time_varying[label].type=random_walk

`mean` Mean
Type: estimable
Default: 0

`sigma` Standard deviation
Type: estimable
Default: 1

`upper_bound` Upper bound for the random walk

Type: constant
Default: 1

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

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

`distribution` distribution
Type: std::string
Default: normal

9.6. Derived quantities

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

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

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

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

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

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

`selectivity` A label for the selectivity
Type: string vector
Default: No Default

`layer_label` A label for the layer that indicates which cells to calculate abundnace in over

Type: std::string
Default: No Default

9.6.2. @derived_quantity[label].type=biomass

`selectivity` A label for the selectivity
Type: string vector
Default: No Default

`biomass_layer_label` A label for the layer that indicates which cells to calculate biomass over
Type: std::string
Default: No Default

9.6.3. @derived_quantity[label].type=biomass_by_cell

9.6.4. @derived_quantity[label].type=mature_biomass

`biomass_layer_label` A label for the layer that indicates which cells to calculate biomass over
Type: std::string
Default: No Default

9.7. Selectivities

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

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

`type` The type of selectivity
Type: std::string
Default: No Default

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

`include_age_zero` Include 0 aged fish in selectivity (more for comparing with population models that start modelling fish at age = 1
Type: boolean
Default: false

9.7.1. @selectivity[label].type=all_values

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

9.7.2. @selectivity[label].type=all_values_bounded

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

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

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

9.7.3. @selectivity[label].type=constant

`c` C
Type: constant
Default: No Default

9.7.4. @selectivity[label].type=double_exponential

`x0` X0
Type: estimable
Default: No Default

`x1` X1
Type: constant
Default: No Default

`x2` X2
Type: constant
Default: No Default

`y0` Y0
Type: estimable
Default: No Default

`y1` Y1
Type: estimable
Default: No Default

`y2` Y2
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

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

`mu` Mu
Type: estimable
Default: No Default

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

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

`alpha` Alpha
Type: estimable
Default: 1.0

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

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

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

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

`alpha` Alpha
Type: constant
Default: 1.0

9.7.7. @selectivity[label].type=inverse_logistic

`a50` A50
Type: estimable
Default: No Default

`ato95` aTo95
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

9.7.8. @selectivity[label].type=knife_edge

`e` Edge
Type: estimable
Default: No Default

`alpha` Alpha
Type: constant
Default: 1.0

9.7.9. @selectivity[label].type=logistic

`a50` A50
Type: estimable
Default: No Default

`ato95` Ato95
Type: estimable
Default: No Default

`alpha` Alpha
Type: estimable
Default: 1.0

`l` Low if(age , Low) = 0

Type: non-negative integer
Default: No Default

`h` High if(age , High) =alpha
Type: non-negative integer
Default: No Default

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

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

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

`a50` A50
Type: constant
Default: No Default

`ato95` Ato95
Type: constant
Default: No Default

`alpha` Alpha
Type: constant
Default: 1.0

9.8. Preference Functions

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

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

`type` The type of selectivity
Type: std::string
Default: No Default

`alpha` The alpha value
Type: float
Default: 1.0

9.8.1. @preference_function[label].type=double_normal

mu
Type: float
Default: No Default

sigma_l
Type: float
Default: No Default

sigma_r
Type: float
Default: No Default

9.8.2. @preference_function[label].type=inverse_logistic

a50 the a50 parameter of the logistic function
Type: float
Default: No Default

ato95 the ato95 parameter of the logistic function
Type: float
Default: No Default

9.8.3. @preference_function[label].type=logistic

a50 the a50 parameter of the logistic function
Type: float
Default: No Default

ato95 the ato95 parameter of the logistic function
Type: float
Default: No Default

9.8.4. @preference_function[label].type=normal

mu
Type: float
Default: No Default

sigma
Type: float
Default: No Default

9.9. Layers

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

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

`type` The type of layer
Type: string
Default: No Default

9.9.1. `@layer[label].type=numeric`

`table layer`
Table contents here
`end_table`
Type: table
Default: No Default

`proportions` is the table proportions
Type: boolean
Default: false

9.9.2. `@layer[label].type=numeric_meta`

`years` The year to apply the layer in
Type: string
Default: No Default

`default_layer` The layer to apply in initialisation phase
Type: string
Default: No Default

`layer_labels` The labels for corresponding *numeric* layers to apply in the above years
Type: string
Default: No Default

9.9.3. `@layer[label].type=integer`

`table layer`
Table contents here
`end_table`
Type: table
Default: No Default

9.9.4. `@layer[label].type=categorical`

`table layer`
Table contents here
`end_table`
Type: table
Default: No Default

For Categorical layers do not add ” or ’ to make strings.

An example of how to specify a table in the layers for an Int-layer or integer layer you could specify

```
@layer base_layer
type integer
table layer
1 1 1 1
1 1 0 1
1 1 1 1
end_table
```

for a Numeric layer you could have decimal and negative values

```
table layer
2.2 3.2 -23
6.4 -4.5 2.3
end_table
```

10. Observation command and subcommand syntax

10.1. Observation types

The observation types available are,

Observations of proportions of individuals by age class

Observations of proportions of individuals between categories within each age class

Relative and absolute abundance observations

Relative and absolute biomass observations

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

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

`label` Label
Type: std::string
Default: No Default

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

10.1.1. @observation[label].type=age_length

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

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`cell_layer` The layer that indicates what area to summarise observations over.
Type: std::string
Default: No Default

`cells` The cells we want to generate observations for from the layer of cells supplied
Type: string vector
Default: No Default

`number_of_samples` The number of samples to collect from each cell
Type: non-negative integer vector
Default: No Default

`simulation_likelihood` Simulation likelihood to use
Type: std::string
Default: No Default

10.1.2. @observation[label].type=biomass

`catchability` The Catchability multiplier
Type: estimable
Default: // TODO not sure if neccessarystring

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`error_value` The error values of the observed values (note the units depend on the likelihood
Type: constant vector
Default: No Default

`selectivities` Labels of the selectivities
Type: string vector
Default: true

`proportion_through_mortality_block` Proportion through the mortality block of the time step to infer observation with
Type: constant
Default: double(0.5
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`cell_layer` The layer that indicates what area to summarise observations over.
Type: std::string
Default: No Default

`cells` The cells we want to generate observations for from the layer of cells supplied
Type: string vector
Default: No Default

`simulation_likelihood` Simulation likelihood to use
Type: std::string
Default: No Default

`abundance` Is the index biomass (abundance = false) or abundance (abundance = true
Type: boolean
Default: false

10.1.3. `@observation[label].type=mortality_event_composition`

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`ageing_error` Label of ageing error to use
Type: string
Default: none

`process_label` Label of of removal process
Type: string
Default: ""

`fishery_label` Label of of removal process
Type: string
Default: No Default

`normalise` Are the compositions normalised to sum to one
Type: bool
Default: true
Value: Do you want observation as proportions (normalised to sum to one, or as numbers?
Allowed Values: true, false

`composition_type` Is the composition Age or Length
Type: string
Default: age
Value: Do you want age or length
Allowed Values: age, length

`layer_of_stratum_definitions` The layer that indicates what the stratum boundaries are.
Type: string
Default: No Default

`strata_to_include` The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

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

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

`plus_group` Plus group
Type: bool
Default: true

`sexed` You can ask to 'ignore' sex (only option for unsexed model), or generate composition for both sexes
Type: bool
Default: true

`stratum_weighting_method` Method to weight stratum estimates by
Type: string
Default: biomass
Allowed Values: biomass, area, none

`table error_values` The table of error values. Need a row for each year and column for each `strata_to_include`
Default: No default
Value:
Note: See below for example

```
@observation fishery_age_comp

...

strata_to_include single_area
table error_values
year single_area
1900 400
1901 400
end_table
```

10.1.4. @observation[label].type=mortality_event_biomass_age_clusters

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`ageing_error` Label of ageing error to use
Type: std::string
Default: none

`process_label` Label of of removal process
Type: std::string
Default: ""

`fishery_label` Label of of removal process
Type: std::string
Default: No Default

`cluster_cv` CV for randomly selecting clusters
Type: float
Default: No Default

`composition_type`
Type: std::string
Default: proportions
Value: Do you want observation as proportions (normalised to sum to one, or as numbers?
Allowed Values: proportions, numbers

cluster_sigma Standard deviation for the M-H proposal distribution
Type: float
Default: No Default
Lower Bound: 0.0 (exclusive)

age_samples_per_cluster Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

stratum_weighting_method Method to weight stratum estimates by
Type: std::string
Default: biomass
Allowed Values: biomass, area, none

sexed You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a particular sex, either 'male' or 'female
Type: boolean
Default: false

layer_of_stratum_definitions The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

strata_to_include The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.5. @observation[label].type=mortality_event_biomass_clusters

years The years of the observed values
Type: non-negative integer vector
Default: No Default

ageing_error Label of ageing error to use
Type: std::string
Default: none

process_label Label of of removal process
Type: std::string
Default: ""

fishery_label Label of of removal process
Type: std::string
Default: No Default

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

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

`cluster_cv` CV for randomly selecting clusters
Type: float
Default: No Default

`cluster_sigma` Standard deviation for the M-H proposal distribution
Type: float
Default: No Default
Lower Bound: 0.0 (exclusive)

`age_samples_per_cluster` Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

`length_samples_per_cluster` Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

`final_age_protocol` What method do you want to use to calculate final age composition
Type: std::string
Default: age_length_key
Allowed Values: direct_ageing, age_length_key

`age_allocation_method` The method used to allocate aged individuals across the length distribution
Type: std::string
Default: random
Allowed Values: random, equal, proportional

`stratum_weighting_method` Method to weight stratum estimates by
Type: std::string
Default: biomass
Allowed Values: biomass, area, none

`sex` You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a

particular sex, either 'male' or 'female
Type: std::string
Default: ignore
Allowed Values: male, female, ignore

`layer_of_stratum_definitions` The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

`strata_to_include` The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.6. `@observation[label].type=mortality_event_biomass_clusters_original`

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`ageing_error` Label of ageing error to use
Type: std::string
Default: none

`process_label` Label of of removal process
Type: std::string
Default: ""

`fishery_label` Label of of removal process
Type: std::string
Default: No Default

`cluster_cv` CV for randomly selecting clusters
Type: float
Default: No Default

`cluster_attribute` What attribute do you want to link clusters by, either age or length
Type: std::string
Default: age
Allowed Values: age, length

`cluster_correlation_lambda` The probability of being associated to a cluster based on

distrance from attribute
Type: float
Default: No Default
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`age_samples_per_cluster` Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

`length_samples_per_cluster` Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

`minimum_cluster_weight_to_sample` The minimum weight (tonnes) threshold to consider sampling, should be well in the distribution of cluster sizes
Type: float
Default: No Default

`final_age_protocol` What method do you want to use to calculate final age composition
Type: std::string
Default: age_length_key
Allowed Values: direct_ageing, age_length_key

`age_allocation_method` The method used to allocate aged individuals across the length distribution
Type: std::string
Default: random
Allowed Values: random, equal, proportional

`sex` You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a particular sex, either 'male' or 'female
Type: std::string
Default: ignore
Allowed Values: male, female, ignore

`layer_of_stratum_definitions` The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

`strata_to_include` The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.7. @observation[label].type=mortality_event_biomass_length_clusters

years The years of the observed values
Type: non-negative integer vector
Default: No Default

process_label Label of of removal process
Type: std::string
Default: ""

fishery_label Label of of removal process
Type: std::string
Default: No Default

cluster_cv CV for randomly selecting clusters
Type: float
Default: No Default

composition_type
Type: std::string
Default: proportions
Value: Do you want observation as proportions (normalised to sum to one, or as numbers?
Allowed Values: proportions, numbers

cluster_sigma Standard deviation for the M-H proposal distribution
Type: float
Default: No Default
Lower Bound: 0.0 (exclusive)

length_samples_per_cluster Number of age samples available to be aged per cluster
Type: non-negative integer
Default: No Default

stratum_weighting_method Method to weight stratum estimates by
Type: std::string
Default: biomass
Allowed Values: biomass, area, none

sexed You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a particular sex, either 'male' or 'female
Type: boolean
Default: false

layer_of_stratum_definitions The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

strata_to_include The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.8. @observation[label].type=mortality_event_biomass_scaled_age_frequency

years The years of the observed values
Type: non-negative integer vector
Default: No Default

ageing_error Label of ageing error to use
Type: std::string
Default: none

process_label Label of of removal process
Type: std::string
Default: ""

fishery_label Label of of removal process
Type: std::string
Default: No Default

age_allocation_method The method used to allocate aged individuals across the length distribution
Type: std::string
Default: random
Allowed Values: random, equal, proportional

sex You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a particular sex, either 'male' or 'female
Type: std::string
Default: ignore
Allowed Values: male, female, ignore

layer_of_stratum_definitions The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

strata_to_include The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.9. @observation[label].type=mortality_scaled_age_frequency

years The years of the observed values
Type: non-negative integer vector
Default: No Default

ageing_error Label of ageing error to use
Type: std::string
Default: none

process_label Label of of removal process
Type: std::string
Default: ""

age_allocation_method The method used to allocate aged individuals across the length distribution
Type: std::string
Default: random
Allowed Values: random, equal, proportional

layer_of_stratum_definitions The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

strata_to_include The cells which represent individual stratum to be included in the analysis, default is all cells are used from the layer
Type: string vector
Default: true

10.1.10. @observation[label].type=process_removals_by_age

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

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

plus_group Use age plus group
Type: boolean
Default: true

years Years for which there are observations

Type: non-negative integer vector
Default: No Default

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

`process_label` Label of of removal process
Type: std::string
Default: ""

`normalise` Are the compositions normalised to sum to one
Type: boolean
Default: true

`cell_layer` The layer that indicates what area to summarise observations over.
Type: std::string
Default: No Default

`cells` The cells we want to generate observations for from the layer of cells supplied
Type: string vector
Default: No Default

`sex` You can ask to 'ignore' sex (only option for unsexed model), or generate composition for a particular sex, either 'male' or 'female
Type: std::string
Default: ignore
Allowed Values: male, female, ignore

`simulation_likelihood` Simulation likelihood to use
Type: std::string
Default: No Default

10.1.11. @observation[label].type=process_removals_by_length

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

`process_label` Label of of removal process
Type: std::string
Default: ""

`cell_layer` The layer that indicates what area to summarise observations over.

Type: std::string
Default: No Default

`cells` The cells we want to generate observations for from the layer of cells supplied
Type: string vector
Default: No Default

`simulation_likelihood` Simulation likelihood to use
Type: std::string
Default: No Default

10.1.12. `@observation[label].type=proportions_at_age`

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

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

`sexed` Observation split by sex
Type: boolean
Default: false

`normalise` Are the compositions normalised to sum to one
Type: boolean
Default: true

`selectivities` Labels of the selectivities
Type: string vector
Default: No Default

`proportion_through_mortality_block` Proportion through the mortality block of the time step to infer observation with
Type: float
Default: float(0.5
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`plus_group` max age is a plus group
Type: boolean
Default: true

`years` Years for which to calculate an observation

Type: non-negative integer vector
Default: No Default

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

`cell_layer` The layer that indicates what area to summarise observations over.
Type: std::string
Default: No Default

`cells` The cells we want to generate observations for from the layer of cells supplied
Type: string vector
Default: No Default

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

`simulation_likelihood` Simulation likelihood to use
Type: std::string
Default: No Default

10.1.13. `@observation[label].type=tag_recapture_by_age`

`years` The years of the observed values
Type: non-negative integer vector
Default: No Default

`tag_release_year` The years that the tagged fish were released
Type: non-negative integer
Default: No Default

`ageing_error` Label of ageing error to use
Type: std::string
Default: none

`process_label` Label of of removal process
Type: std::string
Default: ""

`release_stratum` Stratum that individuals were released in
Type: std::string
Default: No Default

recapture_stratum Stratum to record recaptures, that were released in this year and in the release stratum
Type: string vector
Default: true

10.1.14. @observation[label].type=tag_recapture_by_length

years The years of the observed values
Type: non-negative integer vector
Default: No Default

tag_release_year The years that the tagged fish were released
Type: non-negative integer
Default: No Default

sexed Seperate observation by sex
Type: boolean
Default: true

process_label Label of of removal process
Type: std::string
Default: ""

layer_of_strata_definitions The layer that indicates what the stratum boundaries are.
Type: std::string
Default: No Default

release_stratum Stratum that individuals were released in
Type: std::string
Default: No Default

recapture_stratum Stratum to record recaptures, that were released in this year and in the release stratum
Type: string vector
Default: true

10.2. Likelihoods

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

10.2.1. @likelihood[label].type=binomial

10.2.2. @likelihood[label].type=binomial_approx

10.2.3. @likelihood[label].type=dirichlet

10.2.4. @likelihood[label].type=log_normal

10.2.5. @likelihood[label].type=log_normal_with_q

10.2.6. @likelihood[label].type=logistic_normal

`label` Label for Logisitic Normal Likelihood
Type: std::string
Default: No Default

`type` Type of likelihood
Type: std::string
Default: No Default

`rho` The auto-correlation parameter ρ
Type: estimable vector
Default: No Default

`sigma` Sigma parameter in the likelihood
Type: estimable
Default: No Default

`arma` Defines if two rho parameters supplied then covar is assumed to have the correlation matrix of an ARMA(1,1) process
Type: boolean
Default: No Default

`bin_labels` If no covariance matrix parameter then list a vector of bin labels that the covariance matrix will be built for, can be ages or lengths.
Type: non-negative integer vector
Default: false

`sexed` Will the observation be split by sex?
Type: boolean
Default: false

`robust` Robustification term for zero observations
Type: boolean
Default: false

`seperate_by_sex` If data is sexed, should the covariance matrix be seperated by sex?

Type: boolean
Default: false

sex_lag if T and data are sexed, then the AR(n) correlation structure ignores sex and sets lag = —i-j—+1, where i and j index the age or length classes in the data. Ignored if data are not sexed.
Type: boolean
Default: false

10.2.7. @likelihood[label].type=multinomial

10.2.8. @likelihood[label].type=normal

10.2.9. @likelihood[label].type=pseudo

11. Report command and subcommand syntax

11.1. Report commands and subcommands

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

label The label for the report
Type: std::string
Default: No Default

type The type of report
Type: std::string
Default: No Default

file_name The File Name if you want this report to be in a seperate file
Type: std::string
Default: ""

write_mode The write mode
Type: std::string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

11.1.1. @report[label].type=age_frequency_by_cell

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

time_step Time Step label

Type: std::string
Default: ""

do_length_frequency Print the report as length frequency not age.
Type: boolean
Default: false

agent_frequency Is the frequency for agents (true) or individuals (false)?
Type: boolean
Default: false

11.1.2. @report[label].type=age_length_matrix_by_cell

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

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

11.1.3. @report[label].type=ageing_error_matrix

ageing_error Ageing Error label
Type: std::string
Default: No Default

11.1.4. @report[label].type=derived_quantity

11.1.5. @report[label].type=initialisation_partition

do_length_frequency Print the report as length frequency not age.
Type: boolean
Default: false

11.1.6. @report[label].type=likelihood

likelihood Likelihood label that is reported
Type: std::string
Default: ""

11.1.7. @report[label].type=model_attributes

11.1.8. @report[label].type=numeric_layer

layer_label The Numeric Layer label that is reported
Type: std::string
Default: ""

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

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

11.1.9. @report[label].type=observation

observation Observation label
Type: std::string
Default: No Default

11.1.10. @report[label].type=preference_function

preference_function_label Preference Function label
Type: std::string
Default: No Default

preference_values Preference values to report the preference function
Type: constant-float vector
Default: No Default

11.1.11. @report[label].type=process

process Process label that is reported
Type: std::string
Default: ""

11.1.12. @report[label].type=selectivity

selectivity Selectivity name
Type: std::string
Default: No Default

11.1.13. @report[label].type=simulated_observation

observation Observation label
Type: std::string
Default: No Default

cells Cells to aggregate the observaton over.
Type: string vector
Default: true

11.1.14. @report[label].type=standard_header

11.1.15. @report[label].type=summarise_agents

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

time_step Time Step label
Type: std::string
Default: No Default

number_of_agents Number of agents to summarise
Type: non-negative integer
Default: No Default
Lower Bound: 1 (inclusive)

11.1.16. @report[label].type=tagging_info

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

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

tag_release_year Release year
Type: non-negative integer vector
Default: ""

release_cell release cell, for example row 1 and cell 1 = '1-1'

Type: string vector
Default: ””

11.1.17. `@report[label].type=time_varying`

11.1.18. `@report[label].type=world_age_frequency`

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

`time_step` Time Step label
Type: std::string
Default: ””

12. Including commands from other files

`@include` *`file`* Include an external file

`file` The name of the external file to include
Type: string
Default: No default
Value: A valid external file
Condition: The file name must be enclosed in double quotes
Example: `!include "my_file.ibm"`
Note: `!include` does not denote the end of the previous command block as is the case for all other commands

13. Investigating undefined behaviour in the IBM

Whilst the model is in development (This could be forever haha) some classes will be left behind which might leave bugs when reusing these. If a model run quits/exits middway through a run with no error message. The first thing you want to do is use the internal logging system. This involves the following command `ibm -r --loglevel fine > run.log 2> log.out`. This will pipe out the logging info into the file `log.out`. I have found in the past that some undefined model behaviour will still work when the logging is on. If you can't isolate where the error occurs or when you use the logging system the undefined behaviour doesn't repeat. Then we suggest using a C++ debugger as it will almost certainly be an out of memory issue.

13.1. Using a C++ debugger

Compile the code base with debug setting `doBuild.bat debug`. When this has been successfully compiled copy the model configuration files into the executable location `BuildSystem\bin\windows\debug`. Then use gdbsource open a command prompt and use the command prompt `gdb ibm` this will load ibm into gdbsource. Then type `run -r > run.log`. Because this is not optimised during compilation so I suggest decreasing the number of agents significantly. For more info on using gdb see here

14. Post processing output using R

Hopefully when you get the bundle for this program there will be an **R** package, this section describes how you can use that package to read in and view output from CABM. A list of the **R** package functions are below.

- `extract.run()` imports the model's output into the **R** environment.
- `extract.ibm.file()` imports the model's configuration file into the **R** environment as a list where you can change inputs. Useful in simulation based studies
- `write.ibm.file()` prints the a list object (that has the same characteristics as the `extract.ibm.file()`) to a text file.
- `reformat.compositional.data()` converts CABM compositional observation reports into a more traditional matrix.
- `plot.derived_quantities()` plot or extract derived quantities from a model run.
- `create_ibm_layer` A function that creates a file consists of `@layer` block with a user input matrix, useful function for setting up highly spatial models.
- `csl2_bio` turn a ibm biomass observation into a casal2 style observation
- `csl2_fishery_age` turn a ibm fishery age observation into a casal2 style observation
- `csl2_prop_age` turn a ibm proportions at age observation into a casal2 style observation
- `ssb_multiple_runs` A plotting function to plot multiple Derived Quantities in ggplot style
- `plot_numeric_layer` a plotting function that generates a snapshot of biomass from a `numeric_layer` report. It generated the Figure **??**
- `plot_overall_age_freq` A plotting function to plot age frequency at a point in time of the world vs the fishery

Comparing different initialisation starts, as mentioned in the Tips section 4.12 we highly recommend that you reduce the initialisation phase as much as possible. Below is some **R** code that I often use when looking at the effects of different initialisation phase burn-ins.

```
library(ibm) ## for extracting
library(ggplot2) ## for plotting
library(reshape2) ## for reshaping data so its ggplot friendly

## read in a reported output from a ibm-r runs
## --------------------------------
## An important note before running the models below.
## if you do not include the following report in your configuration files
##
## @report init_2
## type initialisation_partition
##
## this code will not work
## --------------------------------
ibm_30 = extract.run("output_30.log")
ibm_50 = extract.run("output_50.log")
ibm_80 = extract.run("output_80.log")
```

```
ibm_120 = extract.run("output_120.log")

names(ibm_50)
mat = ibm_50$init_2$`1`$values

temp = rbind(ibm_30$init_2$`1`$values,ibm_50$init_2$`1`$values, ibm_80$init_2$`1`$values,
    ibm_120$init_2$`1`$values)
temp$burnin = c(rep("30", nrow(mat)),rep("50", nrow(mat)), rep("80", nrow(mat)), rep("120",
    nrow(mat)))

merged = melt(temp)
colnames(merged) = c("cell", "burnin", "age", "frequency")

## plot age frequency for each row of spatial gird
for (i in 1:5) {
  temp_data = merged[substring(merged$`cell`,0,1) == as.character(i),]
  p <- ggplot(temp_data, aes(x = age, y = frequency, color = burnin)) + geom_point()
  p + facet_grid(cell ~ ., scales="free_y")
  print(p)
    Sys.sleep(2);
}
```

This should generate figures like in Figure 14.1

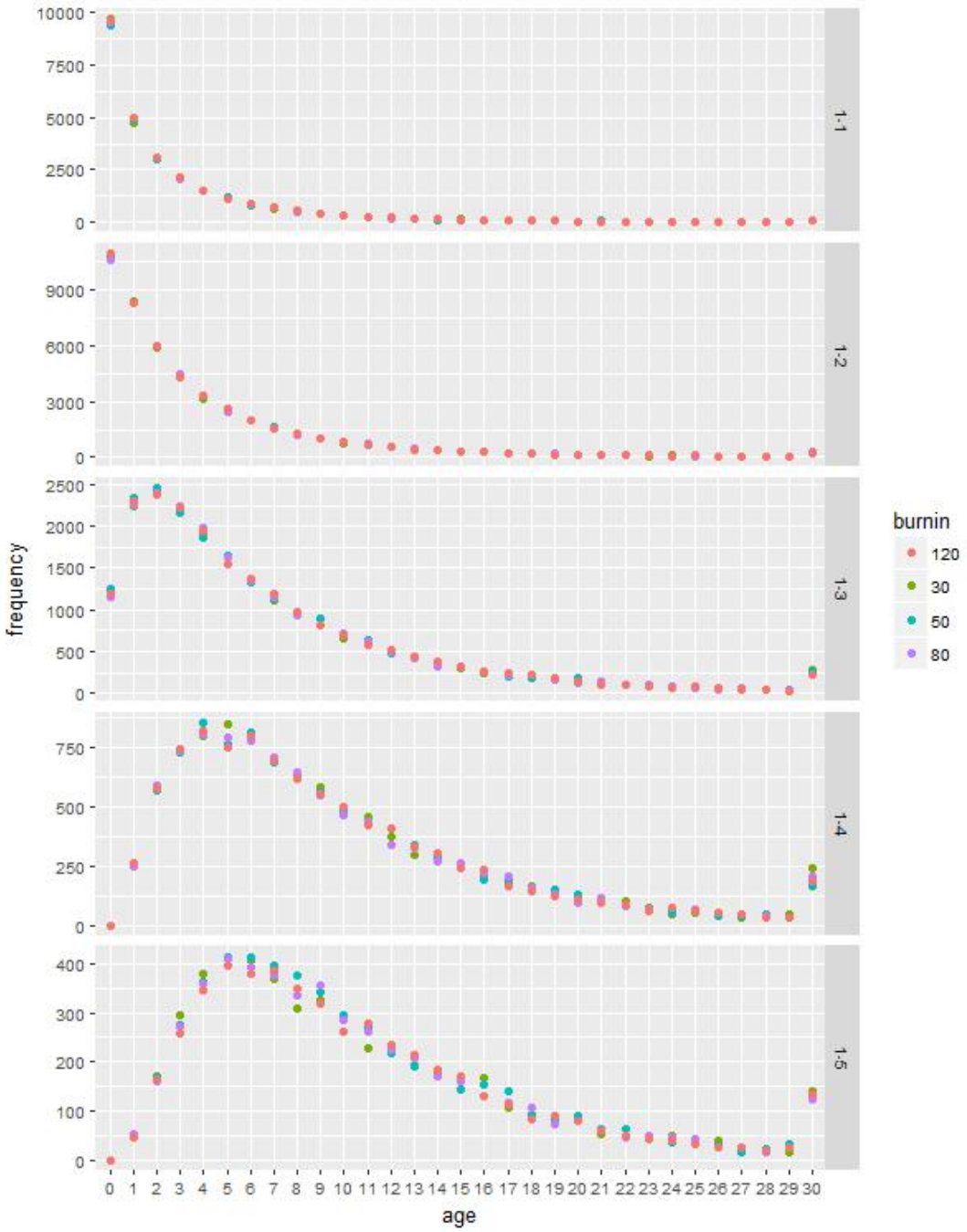


Figure 14.1: An example of plot that looks at the age frequency of different initial burn-in periods

15. Syntax conventions, examples and niceties

15.1. Input File Specification

The file format used for CABM is based on the formats used for Casal2, CASAL and SPM. It's a standard text file that contains definitions organised into blocks.

Without exception, every object specified in a configuration file is part of a block. At the top level blocks have a one-to-one relationships with components in the system.

Some general notes about writing configuration files:

1. Whitespace can be used freely. Tabs and spaces are both accepted
2. A block ends only at the beginning of a new block or end of final configuration file
3. You can include another configuration file from anywhere
4. Included files are placed inline, so you can continue a block in a new file
5. The configuration files support inline declarations of objects

15.1.1. Keywords And Reserved Characters

In order to allow efficient creation of input files ibms file format contains special keywords and characters that cannot be used for labels etc.

@Block Definitions

Every new block in the configuration file must start with a block definition character. The reserved character for this is the `@`character
Example:

```
@block1 <label>
type <type>

@block2 <label>
type <type>
```

'type' Keyword

The 'type' keyword is used for declaring the sub-type of a defined block. Any block object that has multiple sub-types will use the type keyword.
Example:

```
@block1 <label>
type <sub_type>

@block2 <label>
type <sub_type>
```

(Single-Line Comment)

Comments are supported in the configuration file in either single-line (to end-of-line) or multi-line
Example:

```
@block <label>
type <sub_type> #Descriptive comment
#parameter <value_1> { This whole line is commented out
parameter <value_1> #<value_2>(value_2 is commented out)
```

/* */ (Multi-Line Comment)

Multiple line comments are supported by surrounding the comments in /* and */
Example:

```
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>

\*
Do not load this process
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>
*\
```

{} (Indexing Parameters)

Users can reference individual elements of a map using the { } syntax, for example when estimating `ycs_values` you may only want to estimate a block of YCS not all of them say between 1975 and 2012. Example:

```
@time_varying YCS
parameter process[Recruitment].ycs_values{1975:2012}
type constant
value 1
```

':' (Range Specifier)

The range specifier allows you to specify a range of values at once instead of having to input them manually. Ranges can be either incremental or decremental.
Example:

```
@process my_recruitment_process
type constant_recruitment
years_to_run 1999:2009 #With range specifier

@process my_mortality_process
type natural_mortality
years_to_run 2000 2001 2002 2003 2004 2005 2006 2007 #Without range specifier
```

'table' and 'end_table' Keyword

The table keyword is used to define a table of information used as a parameter. The line following the table declaration must contain a list of columns to be used. Following lines are rows of the table. Each row must have the same number of values as the number of columns specified. The table definition must end with the 'end_table' keyword on it's own line. The first row of a table will be the name of the columns if required.
Example:

```
@block <label>
type <sub_type>
parameter <value_1>
table <table_label>
<column_1> <column_2> <column_n>
<row1_value1> <row1_value2> <row1_valueN>
<row2_value1> <row2_value2> <row2_valueN>
end_table
```

[ ] (Inline Declarations)

When an object takes the label of a target object as a parameter this can be replaced with an inline declaration. An inline declaration is a complete declaration of an object one 1 line. This is designed to allow the configuration writer to simplify the configuration writing process.
Example:

```
#With inline declaration with label specified for time step
@model
time_steps step_one=[type=iterative; processes=recruitment ageing]

#With inline declaration with default label (model.1)
@model
time_steps [type=iterative; processes=recruitment ageing]

#Without inline declaration
@model
time_steps step_one

@time_step step_one
processes recruitment ageing
```

Parameters

This syntax can be applied to parameters that are of type map as well, for information on what type a parameter is see the syntax section. An example of a parameter that is of type map is `@time_varying[label].type=constant`. For the following `@time_varying` block,

```
@time_varying q_step1
type constant
parameter catchability[Fishq].q
years  1992 1993 1994 1995
value  0.2 0.2 0.2 0.2
```

In line declaration

In line declarations can help shorten models by passing `@` blocks, for example

```
@observation chatCPUE
type biomass
catchability [q=6.52606e-005]
...
```

In the above code we are defining and estimating catchability without explicitly creating an `@catchability` block.

When you do an inline declaration the new object will be created with the name of the creator's `label.index` where index will be the word if it's one-nine and the number if it's 10+, for example,

```
@mortality halfm
selectivities [type=constant; c=1]

would create
@selectivity halfm.one
```

15.2. Processes

Processes are special in how they can be defined, all throughout this document we have been referring to specifying a process as follows,

```
@process Recruitment
type recruitment_beverton_holt
```

However for convenience and for file clarity you could equally specify this block as follows,

```
@recruitment Recruitment
type beverton_holt
```

The trick is that you can replace the keyword `process` with the first word of the process type, in the example above this is the `recruitment` this can be away of creating more reader friendly/lay term configuration scripts. More examples follow;

```
@mortality Fishing_and_M
type instantaneous


@movement Migration
type box_transfer
```

15.3. An Example Configuration input

I will add this as the project matures, but I suggest users go and look at the Example models. There are a range of models displaying a range of configurations. Especially the SpatialModel that has 100 cells with quarterly time steps and runs for about 30 years. On my computer that runs in about 3.2 minutes.

16. Troubleshooting

16.1. Introduction

This section is to aid users in debugging models, if you cannot resolve an issue using these guidelines then don't hesitate to contact the development team. To report an issue please follow the format described in Section 16.3. We are hoping that most user errors will be well documented and that CABM will produce informative error messages. In the case where this doesn't happen, there are some quick and easy tactics that users can do to attempt to resolve or at least isolate an error/bug. Using CABM's internal logging out system, this is invoked at the command line with the `--loglevel` parameter followed by one of these arguments; `trace`, `finest`, `fine`, `medium`. An example of implementing logging with trace level at the command line is,

```
ibm -r --loglevel trace > output.log 2> log.out
```

The above command will output CABM normal reports into the file "output.log" where as the `2>` syntax will print the error logged out information into the file "log.out". You should be able to see where CABM is exiting by going to the end of the "log.out" file.

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

taken from line 349, of `Model.cpp`

16.2. Reporting errors

If you find a bug or problem in CABM, please email the `CABM Development Team` submit an issue on the github repository found at `https://github.com/Craig44/IBM/issues`. The latter is preferred as it will automatically document the issue which is better than depending on the development team, who may be forgetful. Please follow the guidelines below, as they will enhance the debugging process which can be quite time consuming.

16.3. Guidelines for reporting a problem with CABM

1. Check to ensure you are using the most recent version of CABM. Its possible that the error or problem you are having may have ready been resolved.
2. Describe the version of CABM are you using? e.g., CABM v2022-05-16 (rev. 503df7db) Microsoft Windows executable". The version is provided by CABM with the following command `ibm -v`.
3. What operating system or environment are you using? e.g., "IBM-PC Intel CPU running Microsoft Windows 10 Enterprise".
4. Give a brief one-line description of the problem, e.g., "a segmentation fault was reported".
5. If the problem is reproducible, please list the exact steps required to cause it, remembering to include the relevant CABM configuration file, other input files, and any out generated. Specify the *exact* command line arguments that were used, e.g., "Using the command `***.-* -*` reports a segmentation fault. The input configuration files are attached."
6. If the problem is not reproducible (only happened once, or occasionally for no apparent reason), please describe the circumstances in which it occurred and the symptoms observed (but note it is much harder to reproduce and hence fix non-reproducible bugs, but if several

reports are made over time that relate to the same thing, then this may help to track down the problem), e.g., "CABM crashed, but I cannot reproduce how I did it. It seemed to be related to a local network crash but I cannot be sure."

7. If the problem causes any error messages to appear, please give the *exact* text displayed, e.g., `segmentation fault (core dumped)`.

8. Remember to attach all relevant input and output files so that the problem can be reproduced (it can be helpful to compress these into a single file e.g. zip file). Without these, it is usually not possible to determine the cause of the problem, and we are unlikely to provide any assistance. Note that it is helpful to be as specific as possible when describing the problem.

17. CABM software license

GNU GENERAL PUBLIC LICENSE

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License.

The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

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

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

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

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

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

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

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

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

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

    a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

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

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

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

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

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

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you

from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

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

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

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

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

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

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

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

NO WARRANTY

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

PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

18. Acknowledgements

We thank the developers of Casal2 (Rasmussen et al., 2016), CASAL (Bull et al., 2012), SNA1 agent based model (Jeremy McKenzie & Nokome Bentley) and SPM (Dunn et al., 2015) for their ideas that led to the development of CABM.

Much of the structure of CABM, equations, and documentation in this manual draw heavily on similar components of the fisheries population model Casal2 (Rasmussen et al., 2016), CASAL (Bull et al., 2012) and the spatial model SPM (Dunn et al., 2015). We thank the authors of Casal2, CASAL and SPM for their permission to use their work as the basis for parts of CABM and allow the use of the definitions, concepts, and documentation.

19. References

M. Bertignac, P. Lehodey, and J. Hampton. A spatial population dynamics simulation model of tropical tunas using a habitat index based on environmental parameters. *Fisheries Oceanography*, 7(3-4):326–334, 1998. ISSN 10546006. doi: 10.1046/j.1365-2419.1998.00065.x.

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.

A. Dunn, S. Rasmussen, and S. Mormede. Spatial population model user manual, spm v1.1-2016-03-04 (rev. 1278). Technical Report 138, National Institute of Water and Atmospheric Research Ltd (NIWA), 2015.

R I C C Francis, V Haist, and B Bull. Assessment of hoki (macruronus novaezelandiae) in 2002 using a new model. *New Zealand Fisheries Assessment Report*, 6, 2003.

Volker Grimm and Steven F Railsback. *Individual-based Modeling and Ecology*. Princeton University Press, 2013.

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

Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation: Special Issue on Uniform Random Number Generation*, 8(1):3–30, January 1998.

Joe Scutt Phillips, Alex Sen Gupta, Inna Senina, Erik van Sebille, Michael Lange, Patrick Lehodey, John Hampton, and Simon Nicol. An individual-based model of skipjack tuna (katsuwonus pelamis) movement in the tropical pacific ocean. *Progress in Oceanography*, 164:63–74, 2018.

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/`.

S. Rasmussen, I. Doonan, A. Dunn, C. Marsh, K. Large, and S. Mormede. Casal2 user manual. Technical Report 139, National Institute of Water and Atmospheric Research Ltd (NIWA), 2016.

Jon Schnute. A versatile growth model with statistically stable parameters. *Canadian Journal of fisheries and aquatic sciences*, 38(9):1128–1140, 1981.

Samuel B Truesdell, Deborah R Hart, and Yong Chen. Effects of unequal capture probability on stock assessment abundance and mortality estimates: an example using the us atlantic sea scallop fishery. *Canadian Journal of Fisheries and Aquatic Sciences*, 74(11):1904–1917, 2017.

Appendices

A. An Appendix