THE IBM User Manual

C. Marsh

2018

THE IBM User Manual (modified 2018-08-01) for use with
ibm-v2018-08-01 (rev. bc0688d)

Citation: C. Marsh (2018). THE IBM User Manual, v2018-08-01 (rev. bc0688d). . 98 p.

Contents

1. Introduction

THE IBM ia a project I am currently working on during my PhD and I hope that it will be useful to some one out in the wide world. THE IBM simulates a generalised individual based model that allows a great deal of choice in specifying the agent dynamics, spatial resolution, timing of events and model outputs. THE IBM is designed for flexibility and is spatially explicit. The term Individual in this document also means agent or super individual, where a modelled entity can represent many entities with identical characteristics. Unfortunately I have used these terms interchangeably through out this document and the code. I am hoping to tidy this up for consistency in the future, apologies if this adds confusion.

THE IBM may not be as fast as specific Individual based models as generality usually comes at a cost of speed, but I hope that is fast enough to do most tasks the users desire. The bonus in generality is if it gets uptake in the community you can have a little more faith in the underlying dynamics, also because of the syntax and error handling it should be easy to set up models than it would be to code your own (always pros and cons).

The time period and annual cycle of THE IBM is completely defined by the user. 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 THE IBM I hope will be when it used in management strategy evaluation and population assessment model investigation.

1.1. Where to get THE IBM

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

Currently you have to compile the code, to get an executable but, the repository contains all the required thirdparty libraries and has been developed for ease of compilation (it is very easy see the github page for information). I am hoping for a beta version of THE IBM in August.

1.2. System requirements

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

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

The program itself requires a few gigs of hard-disk space but output files can consume large amounts of disk space. Depending on the number and type of user output requests, the output could range from a few hundred kilobytes to several hundred megabytes. 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 `ibm` or `ibm.exe` is required to run THE IBM. No other software is required. We do not provide a version for 32-bit operating systems.

THE IBM comes with an **R** (R Core Team, 2014) package to assist in the post processing of THE IBM output. We provide the `ibm` **R** package for importing the THE IBM output into **R** (see Section 12).

1.4. Getting help

THE IBM is distributed as unsupported software. The Development Team would appreciate being notified of any problems or errors in THE IBM, please use the github page to post issues, see Section 13.2 for the recommended template for reporting issues.

1.5. Technical details

THE IBM 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 THE IBM 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 THE IBM

I still have 2.5 years of my PhD to go so I am hoping to maintain and continue development during that period. Things that are not in the Beta version that would be great to add one day (The dream-list)

- 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 nigly.
- 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.

- see used in practice I am hoping that this could be used as an leading 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.

2. Model overview

2.1. Introduction

THE IBM is run from the console window in Microsoft Windows or from a terminal window in Linux. THE IBM gets its information from input configuration files, the default file THE IBM will look for is *input configuration file*, 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 THE IBM. Command line switches tell THE IBM 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 individuals 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.

2.2. The population section

This section discusses how to set up the process model, which controls how individuals 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 talks about what observations THE IBM can create and then simulate from. This give likelihood information and syntax examples.

2.4. The report section

This section 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 THE IBM

THE IBM is run from the console window (i.e., the command line) on Microsoft Windows or from a terminal window on Linux. THE IBM 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 THE IBM 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 THE IBM

To use THE IBM, 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 `ibm` with any arguments (see Section 3.4 for the the list of possible arguments) to start your THE IBM job running. THE IBM will print output to the screen and return you to the command prompt when the job has completed. Note that the THE IBM 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 THE IBM installer (**doesn't exist**) 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

The input configuration file is made up of four 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 THE IBM 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 10 for more detail.

3.3. Redirecting standard output

THE IBM uses the `standard output` stream to display run-time information. The standard error stream is used by THE IBM 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

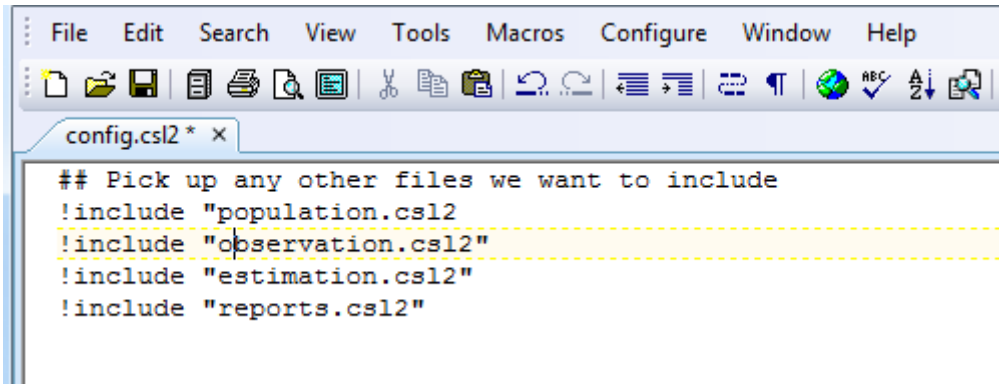


Figure 3.1: Example of using the input configuration file command !*include file*.

systems), you can do this using the command structure,

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

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

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

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

```
ibm [arguments] > out
```

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

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

Note that THE IBM 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 THE IBM 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 THE IBM used to run the model.

3.4. Command line arguments

The call to THE IBM is of the following form:

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

where,

`-c config_file` Define the input configuration file for THE IBM (if omitted, THE IBM 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)

-v [--version] Display the THE IBM version number

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

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 THE IBM will generate a random number seed based on the computer clock time)

--loglevel arg = {trace, finest, fine, medium} (see Section 6)

3.5. Constructing the THE IBM 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 5) contains all the observations data available to the model and describes how the observed values should be formatted, how THE IBM calculates the expected values, and the likelihoods available for each type of observation; and

Report input (Section 6) specifies any output required.

The command and subcommand syntax to be used in each of these configuration files are listed in Sections 7 (Population), 8 (Observation) and 9 (Report).

3.5.1. Commands

THE IBM 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

THE IBM 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 10, 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. THE IBM 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 10 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 THE IBM parameter names

When THE IBM processes an input configuration file it translates each command block and each subcommand block into a unique THE IBM 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{male}
```

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

3.6. THE IBM exit status values

Whether THE IBM 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 THE IBM 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 specifies the model of the population dynamics. It describes the model structure (population structure), defines the population processes (e.g., recruitment, migration, and mortality), the selectivities, and associated parameters.

The population section consists of several components, including:

- The population structure;
- Model initialisation (i.e., the state of the partition at the start of the first year);
- The years over which the model runs (i.e., the start and end years of the model)
- The annual cycle (time-steps and processes that are applied in each time-step);
- The 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 THE IBM is represented by an $n_{rows} \times n_{cols}$ grid, with rows $i = 1 \ldots n_{rows}$ and columns $j = 1 \ldots n_{cols}$. Each cell of this matrix records the population structure at that point in space, where the population structure is represented by a list of individuals. Locations where individuals can and cannot potentially be present using a *layer*.

THE IBM implements a single spatial structure, a grid of *square* cells (Figure 4.1). The spatial grid can be of an arbitrary size, 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 THE IBM 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 the locations where the population can and cannot potentially be present (e.g., in a marine model, the locations associated with the sea and not land). These are defined as the cells where the base layer has a value greater than zero. There must be at least one cell in the spatial grid where the population can be present. In addition, the base layer also defines the relative *area* of each spatial cell that is used for density calculations within THE IBM.

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

Hence, the definition of the spatial structure includes;

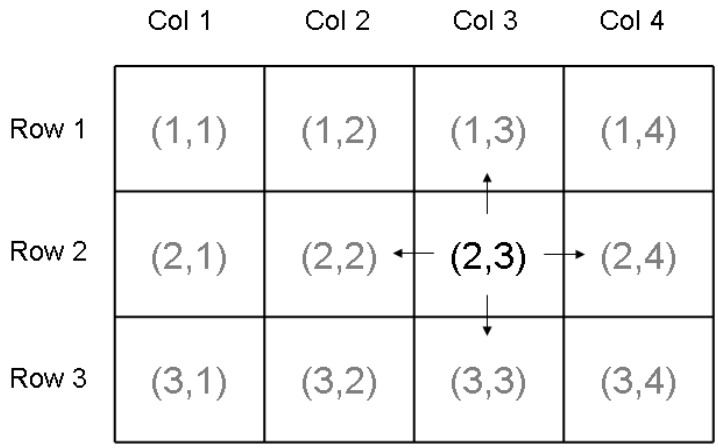


Figure 4.1: An illustration of the spatial structure

- 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
```

See below 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.

4.3. Population structure

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

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

Each year is split into one or more time steps, with at least one process occurring in each time step. Each time step can be thought of as representing a particular part of the calendar year, or 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 removed from the partition (see Section 4.6.2).

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

The state can undergo a number of possible changes, called transitions. Transitions are accomplished by processes, including: recruitment, natural mortality, anthropogenic mortality, ageing, migration, tagging events, and maturation.

The division of the year into an arbitrary number of time steps allows the user to specify the exact order in which processes and observations occur 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.

The key element of the state is the spatial world view, which holds all the entities.

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

```
@model
start_year
final_year
min_age 1
max_age 20
age_plus_group True
initialisation_phases iphase
time_steps step1 step2 step3
```

4.4. Individuals

An individual is the core object in the state that everything manipulates or summarises. An individual has a list of predefined attributes that the user can choose to incorporate in the model or ignore, the list is below.

- age
- length
- weight
- sex
- maturity
- current area, birth area
- lat and long
- population scalar how many other individuals does this represent (scalar)
- natural mortality parameter
- growth parameters
- scalar (if ¿ 1 then this individual represents more than one is stock specific.)

An important attribute, if the model contains super individuals (agents) where a single agent represents many individuals with the same characteristics is the scalar parameter. This parameter is associated with a recruitment event as we use the scalers to scale up a SSB to B_0 for a given recruitment event. This makes the scalar more of a meta-population parameter or a stock parameter

in fisheries. The way a scalar is applied is through the subcommand `recruitment_layer_label` where you link a cell to a recruitment process (for more information see Section 4.6.3). This link says that any individual that was created by that recruitment dynamic gets a stock or recruitment scalar.

Maturity is an attribute that is activated by the maturity process (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.

The other attributes don't need any explanations are usually modified by processes, such as growth, movement and mortality. The attribute that I will spend a bit of time discussing is the method on how we initially set scalars on Individuals.

$$S = \frac{\sum_{s=1}^{n_s} B0_s}{\sum_{s=1}^{n_s} SSB_s} \tag{4.1}$$

Where S is the global scalar that is associated to all new individuals in the system, $B0_s$ is the B_0 parameter for stock s SSB_s is the related spawning stock biomass. There are processes that do modify this, and that usually consists of processes or observations that truly involve a single entity such as tagging, when this occurs adjustments are made to this scalar.

4.5. Layers

Layers are a key underlying concept in THE IBM. 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 THE IBM 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 THE IBM during a model run.

Within THE IBM, 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 the population can and cannot potentially be present (e.g., locations associated with the sea and not land in a marine model). Here, we define that a cell may potentially have part of the population present if every element $L_B(i, j) \geq 0$. 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.

THE IBM 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 a in categories k and with selectivity S_l at age l.

$$N(a) = \sum_k \sum_l S_l \text{ element}(i, j, k, l) \tag{4.2}$$

THE IBM 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}

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

THE IBM 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,

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

THE IBM 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},

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

THE IBM 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: THE IBM 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 THE IBM 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 AnnualNumericLayer
    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 THE IBM.

Mortality blocks are used to define the 'point' in the model time sequence when observations (see Section 5) 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 THE IBM, and cannot be modified. These mortality processes are described in subsection 4.7.4.

THE IBM 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.

THE IBM 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 THE IBM will only calculate the quantity once (either before or after). For example the parameter in the derived quantities class `proportion_through_mortality`.

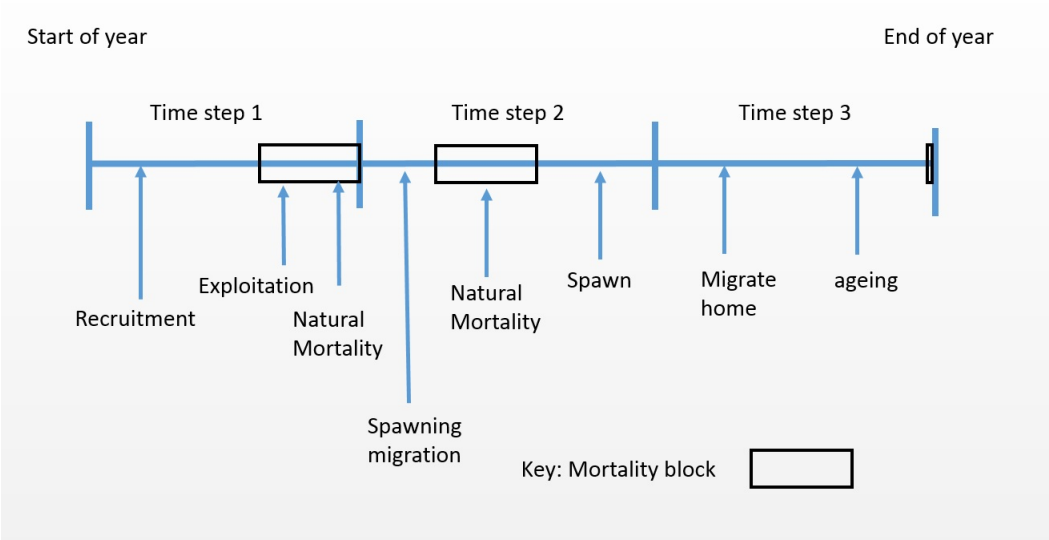


Figure 4.2: A visual representation of a hypothetical sequence for an annual cycle.

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 (e.g exploited), prior to the start year of the model. This can be computationally expensive if a plus group is present in the partition.

Currently users can only initialise the partition via an iterative process. THE IBM does a few tricks to help speed up the initialisation. The first thing THE IBM 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` 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 THE IBM 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 THE IBM 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
number_of_individuals 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. Processes

Processes are a set of classes that get access to agents parameters and either modifies them (growth), moves them (to anther cell), adds new agents (recruitment) or removes them (mortality). Some users may not think having an analagous to population processes fits in the Individual based model framework, but it is structured this way for convenience and code maintenance more than anything. The other thing to note is that most proceses are dictated by agent specific parameters such as natural mortality and growth etc.

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 that, but if you want to account for growth in between annual increments that is possible via time step proportion increments, see growth processes for more information.

4.7.2. Maturity

The process of converting an individual from immature to mature, only should be used if you link it to a mature-biomass derived quantity (Section 4.8). 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 THE IBM 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 individuals 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 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.

4.7.3. Recruitment

Recruitment is the process where new individuals are created in the system. It is also the process that defines stock structure in the model. Stocks are not an explicit attribute of THE IBM 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 parameter and an associated derived quantity that represents the *SSB* for that event.

4.7.3.1. Constant recruitment

4.7.3.2. Beverton-Holt recruitment

4.7.4. Mortality

Mortality processes remove individuals from the model domain (and also from memory).

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

($uniform()$).

$$uniform() \leq 1 - e^{-M_i * S_a} \tag{4.6}$$

where, M_i is the individuals instantaneous mortality rate and S_a is a selectivity at age allowing the process to be age based. M_i the individuals instantaneous mortality rate is generated via the following process

$$M_i \sim LN(M * M_{row,col}, CV) \tag{4.7}$$

where M is the parameter `m` and $M_{row,col}$ is the multiplier layer value in row *row* and column *col*. Users can also set a flag 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 at birth.

An example of how you could set up a constant mortality rate process that varied through time 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
```

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 4.7.4.4). Advantages over using this mortality

process say over the Baranov method (next paragraph 4.7.4.3) is that it allows for individuals to have multiple encounters with a fishery, but the negatives is that you cannot simply add natural mortality processes.

The algorithm

- randomly select an agent, **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 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 fish can interact with fishery many times during a fishing event, but there is no natural mortality.

4.7.4.3. Baranov

This process applies both natural mortality and fishing mortality

The algorithm

- iterate over every individual in the partition or spatial cell
- randomly pick either natural mortality or fishing mortality process to apply, **with replacement**
- check to see if the agent survives this dynamic (compare survivability with a uniform random variable)
- then apply the non-selected dynamic to apply the remaining dynamic (compare survivability with a uniform random variable)
- if agent makes it through it has survived the mortlaity for this time step.

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. 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. THE IBM will then go evaluate the relative effort based on the biomass and soon user defined weighting matrix.

$$E_{row,col} = B_{row,col} * col \tag{4.8}$$

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,

$$P_i^x = f(x_i;\theta) \tag{4.9}$$

where $f(x_i;\theta)$ is a preference function (see Section 4.10). The overall preference in cell i can be denoted as the accumulation of all these preference functions.

$$P_i = \left(\prod_n^{i=1} P_i^x\right)^{1/n} \tag{4.10}$$

where n is the number of preference variables that you want to include 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** and **Y** axis. Currently THE IBM 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

$$u_i = \frac{P_{j,i+h} - P_{j,i-h}}{2} \tag{4.11}$$

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.

$$x_j \sim N(u_i, \sigma_i) \tag{4.12}$$

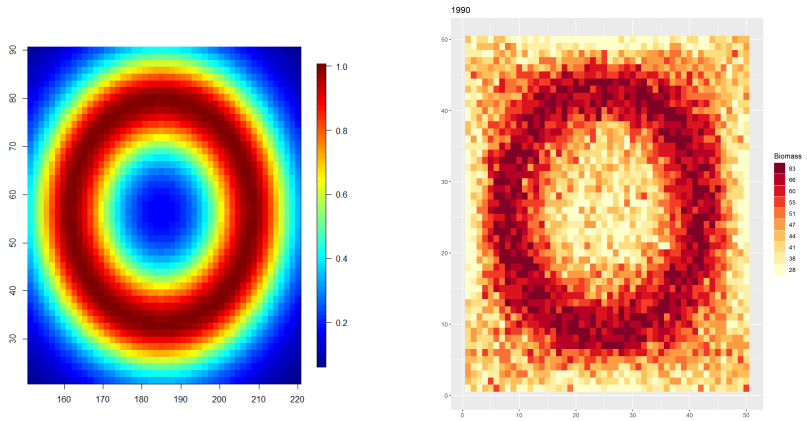
Where u_i is the gradient in the **X** direction, D_i is the diffusion parameter that is scaled based on the habitat preference and x_j is the future location on the X axis of the individual. D_i inversely relates to the quality of the preference habitat and a parameter D_{max}.

$$D_i = D_{max}\left(1 - \frac{P_i}{\zeta + P_i}\right) \tag{4.13}$$

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

$$\sigma_i = \sqrt{2 * D_i * t} \tag{4.14}$$

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

4.8. 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.

Biomass

Mature Biomass

Abundance

4.9. Growth

Growth currently means updating length and weight for an agent, if the Von Bertalanffy formula currently is used (currently the default in agent class for initialising length) it follows the following formula

$$\Delta L_{t,i} = p_t((L_{i,\infty} - L_i)(1 - e^{-k_i})) \tag{4.15}$$

$$L_{t+1,i} = L_{t,i} + \Delta L_{t,i} \tag{4.16}$$

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.

$$\bar{w_i} = a_i L_i^{b_i} \tag{4.17}$$

THE IBM 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.
```

4.10. Preference Functions

4.11. Selectivities

A selectivity is a function that can have a different value for each age class. Selectivities are used throughout THE IBM to interpret observations or to modify the effects of processes on each age class (Section 4). THE IBM 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). THE IBM 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.

4.11.1. constant

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

The constant selectivity has the estimable parameter C.

4.11.2. knife_edge

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

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

4.11.3. all_values

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

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

4.11.4. all_values_bounded

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

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

4.11.5. increasing

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

The increasing ogive has non-estimable parameters L and H. The estimable parameters are π_L, π_{L+1} ... π_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.

4.11.6. `logistic`

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

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

4.11.7. `inverse_logistic`

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

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

4.11.8. `logistic_producing`

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

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

4.11.9. `double_normal`

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

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

4.11.10. `double_exponential`

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

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

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}
```

4.12. Tips setting up configuration files

THE IBM 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.

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 12.

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 run, however the less agents the more coarse you model becomes because then an entity all of a sudden represents 1000 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 observation section

This section describes the observations that THE IBM can generate during run time. THE IBM 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.

5.1. Observations

5.1.1. Process Removals By Age

This observation class aggregates age frequency over a user defined spatial are from a `mortality_event_biomass` 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 process_removals_by_age
cell_layer cells
cells r1-c1 r1-c2 r1-c3 r2-c1 r2-c2 r2-c3 r3-c1 r3-c2 r3-c3
simulation_likelihood multinomial
process_label fishing
years 2000
min_age 0
max_age 28
plus_group true
table error_values
2000 10000
end_table
```

The above syntax asks for an age frequency for each cell in the spatial domain through the link to

the categorical layer. 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 process_removals_by_age
cell_layer cells
cells single_cell
simulation_likelihood multinomial
process_label fishing
years 2000
min_age 0
max_age 28
plus_group true
table error_values
2000 10000
end_table
```

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

5.2. Likelihoods

5.2.1. Likelihoods for proportions-at-age observations

THE IBM implements three likelihoods for proportions-at-age observations, the multinomial likelihood, dirichlet, and the lognormal likelihood.

The multinomial likelihood

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

$$-\log(L) = -\log(N!) + \sum_i \log((NO_i)!) - NO_i \log(Z(E_i, \delta)) \tag{5.1}$$

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

$$Z(\theta, \delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2 - \theta/\delta), & \text{otherwise} \end{cases} \tag{5.2}$$

The default value of δ is 1×10^{-11}.

The dirichlet likelihood

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

$$-\log(L) = -\log(\Gamma\sum_i(\alpha_i)) + \sum_i \log(\Gamma(\alpha_i)) - \sum_i(\alpha_i - 1)\log(Z(O_i,\delta)) \tag{5.3}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.4}$$

The default value of δ is 1×10^{-11}.

The lognormal likelihood

For the observed proportions at age O_i for age classes i, with c.v. c_i, and the expected proportions at the same age classes E_i, the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5\left(\frac{\log(O_i/Z(E_i,\delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{5.5}$$

where

$$\sigma_i = \sqrt{\log\left(1 + c_i^2\right)} \tag{5.6}$$

and the c_i's are the c.v.s for each age class i, and $Z(\theta,\delta)$ is a robustifying function to prevent division by zero errors, with parameter $\delta > 0$. $Z(\theta,\delta)$ is defined as,

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.7}$$

The default value of δ is 1×10^{-11}.

5.2.2. Likelihoods for abundance and biomass observations

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

The lognormal likelihood

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

$$-\log(L) = \sum_i \left(\log(\sigma_i) + 0.5 \left(\frac{\log(O_i/qZ(E_i,\delta))}{\sigma_i} + 0.5\sigma_i \right)^2 \right) \tag{5.8}$$

where

$$\sigma_i = \sqrt{\log\left(1+c_i^2\right)} \tag{5.9}$$

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

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.10}$$

The default value of δ is 1×10^{-11}.

The normal likelihood

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

$$-\log(L) = \sum_i \left(\log(c_i E_i) + 0.5 \left(\frac{O_i - E_i}{Z(c_i E_i, \delta)} \right)^2 \right) \tag{5.11}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.12}$$

The default value of δ is 1×10^{-11}.

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

5.2.3. Likelihoods for tag recapture by age and length observations

The binomial likelihood

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

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

$$-\log(L)' = -\sum_i [\log(n_i!) - \log((n_i - m_i)!) - \log((m_i)!) + m_i \log\left(Z\left(\frac{M_i}{N_i}, \delta\right)\right) + (n_i - m_i)\log\left(Z\left(1 - \frac{M_i}{N_i}, \delta\right)\right)] \tag{5.13}$$

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

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

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

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

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

5.2.4. Likelihoods for proportions-by-category observations

THE IBM implements two likelihoods for proportions-by-category observations, the binomial likelihood, and the normal approximation to the binomial (binomial-approx).

The binomial likelihood

For observed proportions O_i for age class i, where E_i are the expected proportions for age class i, and N_i is the effective sample size for age class i, then the negative log-likelihood is defined as;

$$-\log(L) = -\sum_i [\log(N_i!) - \log((N_i(1 - O_i))!) - \log((N_i O_i)!) + N_i O_i \log(Z(E_i, \delta)) + N_i(1 - O_i)\log(Z(1 - E_i, \delta))] \tag{5.14}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.15}$$

The default value of δ is 1×10^{-11}.

The normal approximation to the binomial likelihood

For observed proportions O_i for age class i, where E_i are the expected proportions for age class i, and N_i is the effective sample size for age class i, then the negative log-likelihood is defined as;

$$-\log(L) = \sum_i \log\left(\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}\right) + \frac{1}{2}\left(\frac{O_i - E_i}{\sqrt{Z(E_i,\delta)Z(1-E_i,\delta)/N_i}}\right)^2 \tag{5.16}$$

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

$$Z(\theta,\delta) = \begin{cases} \theta, & \text{where } \theta \geq r \\ \delta/(2-\theta/\delta), & \text{otherwise} \end{cases} \tag{5.17}$$

The default value of δ is 1×10^{-11}.

6. The report section

The report section specifies the printouts and other outputs from the model. THE IBM 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 THE IBM 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.

6.0.1. 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
```

6.0.2. 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
```

6.0.3. 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
```

6.0.4. 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
```

6.0.5. Derived Quantities

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

```
@report derived_quants
type derived_quantity
```

6.0.6. 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.

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

7. Population command and subcommand syntax

7.1. Model structure

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

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

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

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

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

`age_plus` Define the oldest age 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: true
Value: A list of valid labels defined by `@initialisation_phase`

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

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

`length_plus` Is the last bin a plus group

Type: boolean
Default: false

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

`latitude_bounds` Label for the latitude layer
Type: constant-float vector
Default: true

`longitude_bounds` Label for the longitude layer
Type: constant-float 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 you agent?
Type: boolean
Default: false

`proportion_male` what proportion of the generated agents should be male?
Type: float
Default: 1.0
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

`maturity_ogive_label` Maturity ogive label for each sex
Type: string vector
Default: false

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

`natural_mortality_process_label` Label for the natural mortality process in the annual

cycle
Type: string
Default: No Default

`max_threads_to_use` The maxiumum threads you want to give access to this program
Type: non-negative integer
Default: 1

7.2. Initialisation

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

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

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

7.2.1. `@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

`number_of_individuals` 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: string
Default: ""

7.3. Time-steps

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

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

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

7.4. Processes

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

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

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

7.4.1. @process[label].type=growth_von_bertalanffy_with_basic_growth

`distribution` the distribution to allocate the parameters to the agents
Type: string
Default: No Default

`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: float
Default: No Default

7.4.2. @process[label].type=maturity

7.4.3. @process[label].type=mortality_constant_rate_mortality

7.4.4. @process[label].type=mortality_event_biomass_mortality

7.4.5. @process[label].type=movement_box_transfer_movement

7.4.6. @process[label].type=movement_preference_movement

7.4.7. @process[label].type=nop

7.4.8. @process[label].type=recruitment_beverton_holt

`b0` B0
Type: float
Default: false

`steepness` Steepness
Type: float
Default: 1.0

`ycs_values` YCS (Year-Class-Strength) Values
Type: constant-float vector
Default: No Default

`recruitment_layer_label` A label for the recruitment layer
Type: string
Default: No Default

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

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

`b0` B0
Type: float
Default: false

`recruitment_layer_label` A label for the recruitment layer
Type: string
Default: No Default

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

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

`selectivities`
Type: string vector
Default: No Default

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

7.5. Time varying parameters

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

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

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

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

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

7.5.1. `@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

7.5.2. @time_varying[label].type=constant

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

7.5.3. @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

7.5.4. @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

7.5.5. @time_varying[label].type=random_draw

`mean` Mean
Type: estimable
Default: 0

`sigma` Standard deviation
Type: estimable
Default: 1

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

7.5.6. `@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: string
Default: normal

7.6. Derived quantities

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

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

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

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

`proportion_through_mortality_block` Proportion through the mortality block of the time

step when calculated
Type: float
Default: 0.5
Lower Bound: 0.0 (inclusive)
Upper Bound: 1.0 (inclusive)

7.6.1. @derived_quantity[label].type=abundance

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

`layer_label` A label for the layer that indicates which cells to calculate abundnace in over
Type: string
Default: No Default

7.6.2. @derived_quantity[label].type=biomass

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

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

7.6.3. @derived_quantity[label].type=biomass_by_cell

7.6.4. @derived_quantity[label].type=mature_biomass

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

7.7. Selectivities

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

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

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

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

7.7.1. @selectivity[label].type=all_values

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

7.7.2. @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-float vector
Default: No Default

7.7.3. @selectivity[label].type=constant

`c` C
Type: float
Default: No Default

7.7.4. @selectivity[label].type=double_exponential

`x0` X0
Type: float
Default: No Default

`x1` X1

Type: float
Default: No Default

`x2` X2
Type: float
Default: No Default

`y0` Y0
Type: float
Default: No Default

`y1` Y1
Type: float
Default: No Default

`y2` Y2
Type: float
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

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

`mu` Mu
Type: float
Default: No Default

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

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

`alpha` Alpha
Type: float
Default: 1.0

7.7.6. @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-float vector
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

7.7.7. @selectivity[label].type=inverse_logistic

`a50` A50
Type: float
Default: No Default

`ato95` aTo95
Type: float
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

7.7.8. @selectivity[label].type=knife_edge

`e` Edge
Type: float
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

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

`a50` A50
Type: float
Default: No Default

`ato95` Ato95
Type: float
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

7.7.10. `@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: float
Default: No Default

`ato95` Ato95
Type: float
Default: No Default

`alpha` Alpha
Type: float
Default: 1.0

7.8. 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

7.8.1. @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

7.8.2. @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

7.8.3. @layer[label].type=integer

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

7.8.4. @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
```

8. Observation command and subcommand syntax

8.1. Observation types

The observation types available are,

Observations of proportions of individuals by age class

Observations of proportions of individuals between categories within each age class

Relative and absolute abundance observations

Relative and absolute biomass observations

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

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

`label` Label
Type: string
Default: No Default

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

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

8.1.1. @observation[label].type=age_length

time_step The label of time-step that the observation occurs in
Type: 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: 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

8.1.2. @observation[label].type=biomass

catchability The Catchability to use to scale the observation for vulnerability
Type: string
Default: No Default

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

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

error_value The error values of the observed values (note the units depend on the likelihood
Type: constant-float 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: float
Default: 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: 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

8.1.3. `@observation[label].type=process_removals_by_age`

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

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

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

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

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

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

`cell_layer` The layer that indicates what area to summarise observations over.

Type: 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

8.1.4. `@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: string
Default: ""

`cell_layer` The layer that indicates what area to summarise observations over.
Type: 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

8.2. Likelihoods

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

8.2.1. @likelihood[label].type=binomial

8.2.2. @likelihood[label].type=binomial_approx

8.2.3. @likelihood[label].type=dirichlet

8.2.4. @likelihood[label].type=log_normal

8.2.5. @likelihood[label].type=log_normal_with_q

8.2.6. @likelihood[label].type=multinomial

8.2.7. @likelihood[label].type=normal

8.2.8. @likelihood[label].type=pseudo

9. Report command and subcommand syntax

9.1. Report commands and subcommands

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

label The label for the report
Type: string
Default: No Default

type The type of report
Type: string
Default: No Default

file_name The File Name if you want this report to be in a seperate file
Type: string
Default: ""

write_mode The write mode
Type: string
Default: overwrite
Allowed Values: overwrite, append, incremental_suffix

9.1.1. @report[label].type=age_frequency_by_cell

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

time_step Time Step label

Type: string
Default: ""

9.1.2. @report[label].type=derived_quantity

9.1.3. @report[label].type=initialisation_partition

9.1.4. @report[label].type=model_attributes

9.1.5. @report[label].type=numeric_layer

`layer_label` The Numeric Layer label that is reported
Type: string
Default: ""

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

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

9.1.6. @report[label].type=observation

`observation` Observation label
Type: string
Default: No Default

9.1.7. @report[label].type=process

`process` Process label that is reported
Type: string
Default: ""

9.1.8. @report[label].type=standard_header

9.1.9. @report[label].type=summarise_agents

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

`time_step` Time Step label
Type: string
Default: No Default

`number_of_individuals` Number of agents to summarise
Type: non-negative integer
Default: No Default
Lower Bound: 1 (inclusive)

9.1.10. `@report[label].type=world_age_frequency`

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

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

10. 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

11. Syntax conventions, examples and niceties

11.1. Input File Specification

The file format used for THE IBM 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

11.1.1. Keywords And Reserved Characters

In order to allow efficient creation of input files CASAL2's file format contains special keywords and characters that cannot be used for labels etc.

@Block Definitions

Every new block in the configuration file must start with a block definition character. The reserved character for this is the `@`character
Example:

```
@block1 <label>
type <type>

@block2 <label>
type <type>
```

'type' Keyword

The 'type' keyword is used for declaring the sub-type of a defined block. Any block object that has multiple sub-types will use the type keyword.
Example:

```
@block1 <label>
type <sub_type>

@block2 <label>
type <sub_type>
```

(Single-Line Comment)

Comments are supported in the configuration file in either single-line (to end-of-line) or multi-line
Example:

```
@block <label>
type <sub_type> #Descriptive comment
#parameter <value_1>  This whole line is commented out
parameter <value_1> #<value_2>(value_2 is commented out)
```

/* */ (Multi-Line Comment)

Multiple line comments are supported by surrounding the comments in /* and */
Example:

```
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>

\*
Do not load this process
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>
*\
```

{} (Indexing Parameters)

Users can reference individual elements of a map using the { } syntax, for example when estimating `ycs_values` you may only want to estimate a block of YCS not all of them say between 1975 and 2012. Example:

```
@estimate YCS
parameter process[Recruitment].ycs_values{1975:2012}
type uniform
lower_bound
upper_bound
```

':' (Range Specifier)

The range specifier allows you to specify a range of values at once instead of having to input them manually. Ranges can be either incremental or decremental.
Example:

```
@process my_recruitment_process
type constant_recruitment
years_to_run 1999:2009 #With range specifier

@process my_mortality_process
type natural_mortality
years_to_run 2000 2001 2002 2003 2004 2005 2006 2007 #Without range specifier
```

'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

THE IBM also allows parameters that are of type vector or map to be referenced and estimated partially. An example of a parameter that is type vector is `ycs_values` in a recruitment process. Let say a recruitment block was specified as follows,

```
@process WestRecruitment
type recruitment_beverton_holt
r0 400000
years
ycs_values 1 1 1 1 1 1 1 1
ycs_years 1975:1983
An alternative specification to the sequence of values you can use an astrix to
shorthand repeating integers e.g.
```

```
ycs_values 1*8

steepness 0.9
age 1
```

Lets say we wanted to only estimate the last four years of the parameter `process[WestRecruitment].ycs_values`. This can be done as specified in the following `@estimate` block,

```
@estimate
parameter process[WestRecruitment].ycs_values{1979:1983}
type uniform
lower_bound 0.1 0.1 0.1 0.1
upper_bound  10  10  10  10
```

Note the first element of a vector is indexed by 1. This syntax can be applied to parameters that are of type map as well, for information on what type a parameter is see the syntax section. An example of a parameter that is of type map is `@time_varying[label].type=constant`. For the following `@time_varying` block,

```
@time_varying q_step1
type constant
parameter catchability[Fishq].q
years  1992 1993 1994 1995
value  0.2 0.2 0.2 0.2
```

In this example a user may want to estimate only one element of the map (say 1992), but force all other years to be the same as the one estimate. This can be done in an estimate block as follows,

```
@estimate
parameter time_varying[q_step1].value{1992}
same time_varying[q_step1].value{1993:1995}
type uniform
lower_bound 0.1 0.1 0.1 0.1
upper_bound  10  10  10  10
```

In line declaration

In line declarations can help shorten models by passing `@` blocks, for example

```
@observation chatCPUE
type biomass
catchability [q=6.52606e-005]
time_step one
categories male+female
selectivities chatFselMale chatFselFemale
likelihood lognormal
years 1992:2001
time_step_proportion 1.0
obs 1.50 1.10 0.93 1.33 1.53 0.90 0.68 0.75 0.57 1.23
error_value 0.35

@estimate
```

```
parameter catchability[chatTANbiomass.one].q
type uniform_log
lower_bound 1e-2
upper_bound 1
In line declaration tips
```

In the above code we are defining and estimating catchability without explicitly creating an `@catchability` block.

When you do an inline declaration the new object will be created with the name of the creator's `label.index` where index will be the word if it's one-nine and the number if it's 10+, for example,

```
@mortality halfm
selectivities [type=constant; c=1]

would create
@selectivity halfm.one
```

if there were 10 categories all with there own selectivity the 10^th selectivity would be labelled,

```
@selectivity halfm.10
```

11.2. 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

@transition Migration
type category
```

12. 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 THE IBM. 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
- `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 THE IBM compositional observation reports into a more traditional matrix.
- `plot.derived_quantities()` plot or extract derived quantities from a model run.

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 12.1

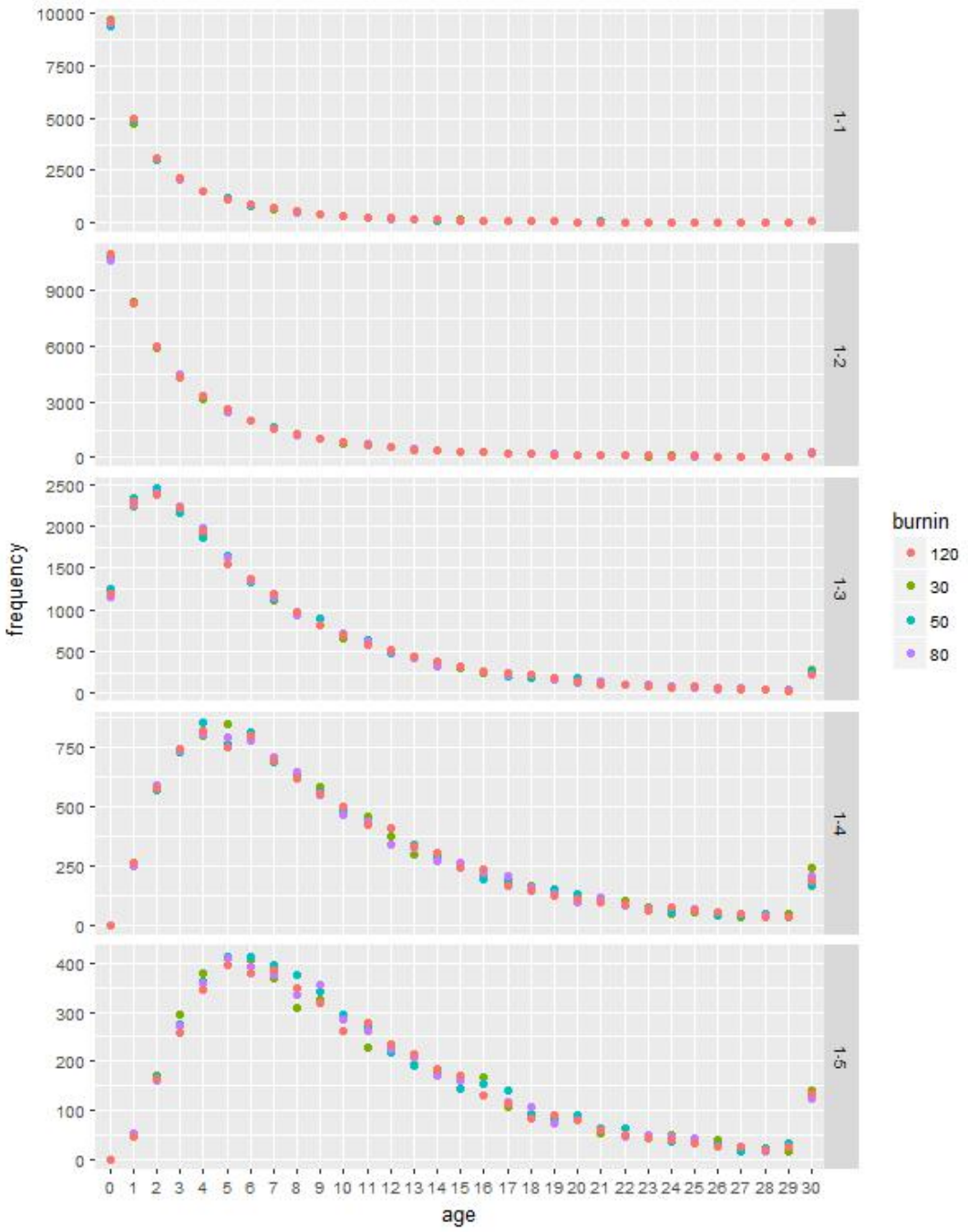


Figure 12.1: An example of plot that looks at the age frequency of different initial burn-in periods

13. Troubleshooting

13.1. 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 13.3. We are hoping that most user errors will be well documented and that THE IBM 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 THE IBM'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 THE IBM 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 THE IBM 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`

13.2. Reporting errors

If you find a bug or problem in THE IBM, please email the `IBM 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.

13.3. Guidelines for reporting a problem with THE IBM

1. Check to ensure you are using the most recent version of THE IBM. Its possible that the error or problem you are having may have ready been resolved.

2. Describe the version of THE IBM are you using? e.g., THE IBM v2018-08-01 (rev. bc0688d) Microsoft Windows executable". The version is provided by THE IBM 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 THE IBM 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., "THE IBM 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.

14. THE IBM 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.

15. Acknowledgements

We thank the developers of Casal2 (Rasmussen et al., 2016), CASAL (Bull et al., 2012) and (Dunn et al., 2015) for their ideas that led to the development of THE IBM.

Much of the structure of THE IBM, 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 THE IBM and allow the use of the definitions, concepts, and documentation.

16. 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.

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.

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

17. Index