
OSMOSE Documentation

Release 4.3

**Nicolas Barrier
Yunne-Jai Shin
Philippe Verley
Morgane Travers
Laure Velez
Ricardo Oliveros-Ramos
Arnaud Grüss
Alaia Morell**

Dec 08, 2021

CONTENTS:

1	Getting started	3
2	Osmose model	9
3	Pre/post processing of OSMOSE model	53
4	Running a calibration	67
5	Bibliography	83
6	Miscellaneous	85
7	Parameter index	89
	Bibliography	91
	Index	93



The present document contains all the necessary information and guidelines to help understand the principles and assumptions of the OSMOSE model, apply it to a specific case study and run simulations for addressing specific issues. The OSMOSE model aims at exploring fish community dynamics and the ecosystem effects of fishing and climate change. It is an Individual-based model (IBM) which focuses on fish species and their trophic interactions ([SC01, SC04]).

The Osmose model assumes opportunistic predation based on spatial co-occurrence and size adequacy between a predator and its prey (size-based opportunistic predation). It represents fish individuals grouped in schools, which are characterized by their size, weight, age, taxonomy and geographical location (2D model), and which undergo different processes of fish life cycle (growth, explicit predation, natural and starvation mortalities, reproduction and migration) and a fishing mortality distinct for each species and structured by age/size, space and season. The model needs basic biological parameters for growth and reproduction processes, that are often available for a wide range of species, and which can be found in FishBase for instance. It also needs to be forced by spatial distribution maps for each species, by age/size/stage and by season depending on data availability. In output, a variety of size-based and species-based ecological indicators can be produced and compared to in situ data (surveys and catch data) at different levels of aggregation: at the species level (e.g. mean size, mean size-at-age, maximum size, mean trophic level, within-species distribution of TL), and at the community level (e.g. slope and intercept of size spectrum, Shannon diversity index, mean TL of catch). The model can be fitted to observed biomass and catch data, using a dedicated evolutionary algorithm. Recent developments have focused on the coupling of OSMOSE to various hydrodynamic and biogeochemical models, allowing to build end-to-end models of marine ecosystems that explicit combined effects of climate and fishing on fish dynamics.

GETTING STARTED

In this section, download and install instructions are provided.

1.1 Setup of the Osmose environment

1.1.1 Download Java

Since Osmose numerical JAVA core is coded in JAVA, Java need to be installed. Beforehand, let us clarify some of the acronyms regarding the Java technologies.

JVM: Java Virtual Machine. It is a set of software programs that interprets the Java byte code.

JRE: Java Runtime Environment. It is a kit distributed by Sun to execute Java programs. A JRE provides a JVM and some basic Java libraries. **A JRE is needed to run Osmose.** It can be downloaded from <https://www.java.com/fr/download/>.

JDK or SDK: Java (or Software) Development Kit bound to the programmer. It provides a JRE, a compiler, useful programs, examples and the source of the API (Application Programming Interface: some standard libraries). **A JDK is needed in order to modify the Osmose Java code.**

1.1.2 Download R

The Osmose-Java core is embedded in an Osmose-R package, which allows to pre-process, run and post-process Osmose outputs. Therefore, it is strongly advised that R be installed. Download instructions are available for [Windows](#), [Linux](#) and [Mac Os X](#).

It is also recommended to install the RStudio GUI (<https://rstudio.com/>).

1.1.3 Defining the Osmose target directory

Since Osmose version 3.3.4, Java executables and demo configuration files have been moved out of the R build to meet CRAN requirements on size package. These files are now downloaded from the Internet and moved to a local directory.

By default, a temporary directory is used; but in this case, the Java code will be downloaded at each new session. To define a directory where to put these downloads, the user need to edit or create a `~/Renvirom` file and to define the `OSMOSE_DIR` environment variable. More defined can be found on the [Osmose-R CRAN page](#)

1.2 Installing Osmose

The Osmose model is provided as a combination of a Java numerical core (referred to as Osmose-Java), associated with an Osmose R package (referred to as Osmose-R). The code is stored in two GitHub directories, a public one (<https://github.com/osmose-model/osmose>) and a private one (for developers only, <https://github.com/osmose-model/osmose-private>). There are several ways to install the Osmose model.

1.2.1 CRAN

The Osmose version on the `master` branch of the public repository is consistent with the most recent version of the package that is submitted to the CRAN. To install this version, open a R session and type `install.packages("osmose")`

1.2.2 Manual install

In order to use the development version of the Osmose model, it must be installed manually

Install from source files

The first way is to clone or download the source code and to install the code manually. To clone the directory, type in a Terminal:

To clone the Osmose repository:

```
# using HTTPS:
git clone https://github.com/osmose-model/osmose-private.git

# using SSH
git clone git@github.com:osmose-model/osmose-private.git
```

When a new version of the code is released, it can be updated as follows:

```
git pull
```

Note: When using the SSH, it is necessary to generate a RSA key that will connect the computer and the remote repository (see [Github Help](#) for details)

When the code has been downloaded, it must be installed as follows:

```
R CMD INSTALL osmose
```

Warning: The code must be reinstalled after each upgrade

Using devtools or RStudio

The Osmose package can also be by using the *devtools* R package or RStudio.

devtools

Open a R session and type the following lines:

```
library("devtools")
install_github("osmose-model/osmose")
```

RStudio

To install Osmose using RStudio, click on the *File* → *New Project* menu and open the *Version Control* → *Git* menu. Set the package URL (<https://github.com/osmose-model/osmose-private.git>). When the project is opened, click on the *Build & Reload* button to install the package.

1.3 Installing NetCDF library (Osmose >= 4.3)

Since Osmose 4.3.0, the Java library that manages input/outputs of NetCDF files requires the external NetCDF C library.

1.3.1 Mac Os X

To install the library on a Mac Os system, open a Terminal and type:

```
sudo port install netcdf4
```

1.3.2 Linux

To install the library on a Linux system, open a Terminal and type:

```
sudo apt-get install netcdf4
```

1.3.3 Windows

To install the library on a Windows system, download the pre-built libraries in [Unidata website](#)

1.4 Code architecture

1.4.1 Osmose <= 4.2.0

For Osmose <= 4.2.0, the directory contains the following folders, related to the R package

- The `inst/java` contains the Osmose Java core, provided as `.jar`.
- The `java` directory contains the Osmose Java source files associated with the `.jar` files.

- The `R` folder contains the R functions (pre/post processing tools, call to the JAVA core, etc.)
- The `data-raw` directory contains input files that can be used to run the model for the first time.
- The `man` and `vignettes` directories contain help files.

1.4.2 Osmose >= 4.3.0

For Osmose >= 4.3.0, the directory also contains the Java source files and compilation tools of the Osmose-Java core:

- The `src` directory contains the Java source files of the current Osmose version.
- The `.xml` files are Maven compilation files.
- The `local` directory contains external Java libraries needed by Osmose

1.5 Installation of the calibration library

The parameters of the Osmose model must be calibrated for each configuration. This is achieved by using the `calibrar` library.

The library is stored in GitHub (<https://github.com/roliveros-ramos/calibrar.git>) and can be installed following the same methods as described in [Section 1.2](#), after changing the URL address.

1.6 (Optional) Compiling Osmose-Java

The compilation of Osmose-Java is not necessary to run Osmose, since Java executables are provided in the package. However, if the user wants to edit and recompile the Osmose-Java core, instructions are provided below.

1.6.1 Netbeans (Osmose <= 4.2.0)

Up to version 4.2.2, the only way to compile the Osmose Java core is by using the integrated development environment (IDE) Netbeans (<https://netbeans.org/downloads/>)

Note: It is highly advised to install together the JDK and the Netbeans bundle: <https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html>

The modification of the Java code is done as follows:

- Unzip one of the `.zip` file of the `osmose/java` directory
- Open Netbeans
- Click on *Open a project*
- Select the folder that has been extracted (should have a coffee cup icon)
- Edit the code
- Clean and build the project by pressing *Maj + F11*

This new `.jar` file can be used in the `run_osmose` function of the Osmose R package.

1.6.2 Maven (Osmose >= 4.3.0)

From version 4.3.0, the code can be compiled independently of the Netbeans IDE by using the Apache Maven software project management and comprehension tool (<https://maven.apache.org/index.html>). When Maven is installed:

- Unzip one of the .zip file of the osmose/java directory
- Navigate to the directory via the Terminal (Linux/Mac) or Cmd (Windows) panel.
- Type `mvn install`

The sources will be built in the *target* directory.

Note: Maven projects can also be built with Netbeans, Eclipse or Visual Studio

1.7 Running the test configuration

To run the test configuration, launch R and type the following:

```
rm(list=ls())
getwd()

library(osmose)

demo = osmose_demo(path = "./", config="eec_4.3.0")
print(demo$config_file)

# run the osmose java core
run_osmose(demo$config_file, parameters="-force")

# read the osmose outputs
data = read_osmose(demo$output_dir)

# plot the outputs
png(file="astart/_static/osmose_ref_conf.png")
plot(data)
```

You should obtain the following figure:

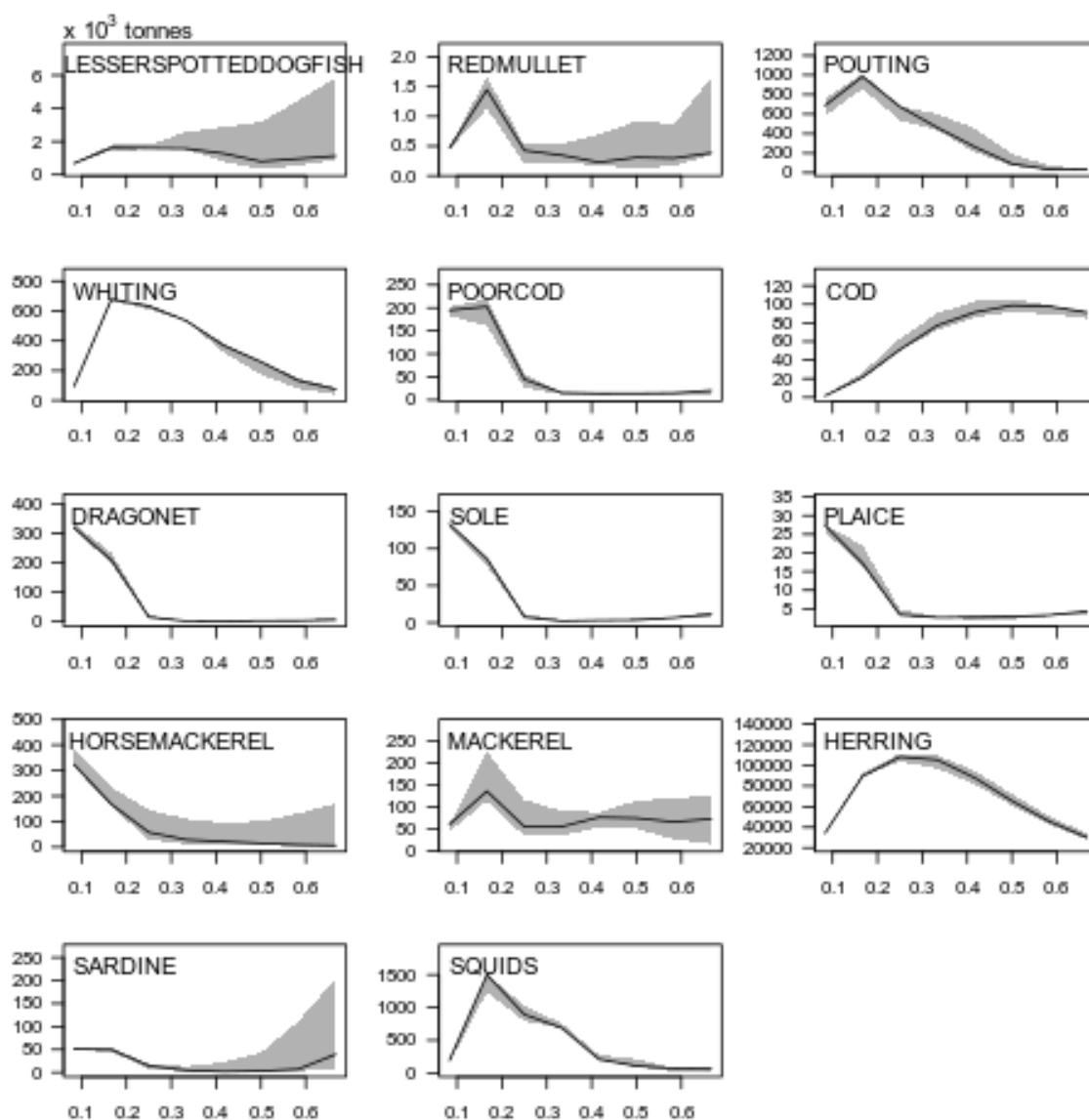


Fig. 1.1: Outputs of the reference configuration

OSMOSE MODEL

In this section, a detailed description of the Osmose model is provided.

2.1 Format and organization of the input files

2.1.1 Configuration files

An Osmose configuration file is a text based file. The name of the file does not matter, but we recommend that you avoid any special characters and space in the name of the file as it might generate IO errors when you'll be running Osmose from the command line or calibrating the model in a UNIX environment. The extension of the file does not matter either.

We call the main configuration file the file that is either listed in Osmose filePath.txt or given as a command line argument. The main configuration file can contain comments, blank lines and parameters. Here is how the Osmose configuration manager proceeds when it opens the main configuration file. It scans every line of the file looking for parameters. Some lines are automatically discarded:

- empty lines (regardless of blank or tab characters).
- lines that start with a punctuation character, one of !"#\$%&'()*+,-./:;<=>?@[]^_`|~

For comments, it is recommended to start the line with # or //

A parameter is formed by the juxtaposition of three elements: **key**, **separator** and **value**.

The **key** can be any sequence of characters, without blank or any special characters (dot, hyphen and underscore are accepted). Example of keys:

```
simulation.ncpu
predation.ingestion.rate.max.sp6
```

Osmose makes no difference between upper and lower case: *simulation.ncpu*, *simulation.Ncpu*, *Simulation.nCPU*, *SIMULATION.NCPU* designate the same key.

Keys starting with *osmose.configuration.** (the star * being any sequence of characters that follow the same rules than any other key) has a special meaning to the configuration manager. It means the value of this parameter is the path of an other Osmose configuration file and the parameters in this file are to be loaded in the current configuration. That way, instead of having one big configuration file with all the parameters, it is possible to split the parameters in as many files as the user wishes. This process works recursively: one file contains one or several parameters *osmose.configuration.** that point to configuration files that may contains one or several parameters *osmose.configuration.**, and so on. OSMOSE handles the sub-configuration file exactly the same way as it handles the main configuration file (same convention for comments, special characters and naming of). As mentioned previously, the **main configuration file** designates the file that is listed in *filePath.txt* or given to Osmose as an input argument.

The separator can be any of the following characters:

- equals =
- semicolon ;
- comma ,
- colon :
- tab \t

Parameters in the same configuration file can have different separators (though it is advisable to be consistent and use the same one). The configuration manager finds out what is the separator for each parameter. The value can be any sequence of characters (even empty). The configuration manager does not try to interpret the value when it loads the configuration files, it merely stores it in a String object. A value can be served by the configuration manager as:

- a string
- an integer
- a float
- a double
- a boolean
- an array of strings, String[]
- an array of integers, int[]
- an array of floats, float[]
- an array of doubles, double[]
- a resolved path

An array of values is a sequence of values with a separator in between: *value1 separator value2 separator value3 separator value4*. Accepted separators for an array of values are the same characters listed above. The separator for an array of values can either be the same or distinct from the separator between the key and the value. The following examples are valid entries:

```
movement.map0.season;0;1;2;3;4;5
movement.map0.season=0;1;2;3;4;5
movement.map0.season = 0, 1, 2, 3, 4, 5
movement.map0.season : 0 ; 1 ; 2;3;4;5
```

and are equivalent for the configuration manager. It can be summarize as:

```
key separator1 value1 separator2 value2 separator2 value3 separator2 value4
```

with separator1 either equal or different from separator2.

2.1.2 CSV input file separator

Many Osmose parameters are paths to CSV file, for instance: .. code-block:: bash

```
movement.map0.file mortality.fishing.rate.byDt.byAge.file.sp# reproduction.season.file.sp#
```

In Osmose 3 and Osmose 3 Update 1 these CSV input files had to be semicolon separated. Since Osmose 3 Update 2, CSV input file separators can be any of the following characters:

- equals =
- semicolon ;
- comma ,
- colon :
- tab \t

Osmose will detect the separator automatically and independently for every CSV file. It means that one CSV input file may be comma separated and an other one may be tab-separated, this is perfectly fine since Osmose 3 Update 2.

2.1.3 Decimal separator

Osmose is quite flexible in terms of separators for the configuration files (automatically detected among = , ; : t), the CSV output files (user-defined by parameter output.csv.separator) and the CSV input files (automatically detected among = , ; : t). On the contrary it restricts the decimal separator to dot, and only dot.

Example given: 3.14159265 or 1.618

Any other decimal separator (COMMA for instance as in French locale) will be misunderstood and will unmistakably lead to errors. One must be careful when editing CSV input files (either parameters or time series) with tools such as spreadsheets that may automatically replace decimal separator depending on the locale settings. Future Osmose release might allow the use of specific locale but for now remember that DOT is the only accepted decimal separator.

2.2 State variables and scales

The basic units of OSMOSE are fish schools, which are composed of individuals that belong to the same species, and that have the same age, size (length, weight), food requirements and, at a given time step, the same geographical coordinates. From the school states (hereafter called individual states), biomass and abundance can be tracked at the population or community levels along with the size, age, and spatial dimensions (Table 2.2).

Other variables can be reported such as the trophic level, the diets, the different sources of mortality, the catches from fishing operations. Because each school simulated in OSMOSE is represented from the egg stage to the terminal age, which necessitates high calculation and memory capacities, and because comprehensive information on entire life cycles needs to be parameterized, the selection of focus species is made parsimoniously, and usually between 10 and 20 high-trophic level species or functional groups are explicitly considered in OSMOSE applications.

The model operates on a weekly to monthly time step, and runs up to 100 years or more depending on applications and simulations.

For eggs (age 0), weight and sizes are provided as parameters. For the others, conversion from size to weight (and conversely) is obtained by using allometric relationships:

$$W = c \times L^b$$

$$L = \left(\frac{W}{c} \right)^{\frac{1}{b}}$$

where the c parameter is a ‘condition.factor’, and b the ‘allometric.power’.

Biomass to abundance conversion for a school is made by using the mean weight of the school:

$$B = N \times W$$

$$N = \frac{B}{W}$$

Table 2.1: Allometric parameters

species.length2weight.condition.factor.sp#	Allometric factor (c)
species.length2weight.allometric.power.sp#	Allometric power (b)
species.egg.size.sp#	Egg size (cm)
species.egg.weight.sp#	Egg weight (g)

Table 2.2: List of state variables

Individual variables	State	Description	Auxiliary state variables / indicators
abundance		Number of fish (N) in the school at the beginning of the time step	
biomass		Biomass (B) of the school at the beginning of the time step (tons)	
age		Age of the fish (year)	species N or B per age class
length		Size of the fish (cm)	fish N or B per size class (size spectrum), mean size of fish, large fish indicator
weight		Weight of the fish (g)	
trophicLevel		Trophic level (TL) of the fish	fish N or B per TL (trophic spectrum), TL of species, TL of catches
nDead[]		Number of dead fish in the current time step for each mortality cause (predation, fishing, natural mortality, starvation)	Catches per species, size class, age class
predSuccessRate		Ingested biomass at current time step/maximum ingestion rate	
preyedBiomass		Biomass of prey ingested by the school at current time step (tons)	fish diets per species, per size class, per age class
lat, lon		location of the fish school in latitude and longitude coordinates	

2.3 Grid

The grid defines the geographical extent of the simulated domain and the spatial discretization.

2.3.1 Osmose <= 4.1.0

In Osmose versions prior to 4.1.0, there were several ways to define the Osmose grid, depending on the input format of resource variables. This was controlled by using the `grid.java.classname` parameter.

`fr.ird.osmose.grid.NcGrid.java`

The easiest way to define an Osmose grid is via a NetCDF file, containing the geographical coordinates and a land/sea mask. It is parameterized as follows:

Table 2.3: Parameters for the `NcGrid.java` class

<code>grid.netcdf.file</code>	Name of the NetCDF file
<code>grid.var.lat</code>	Name of the latitude variable
<code>grid.var.lon</code>	Name of the longitude variable
<code>grid.var.mask</code>	Name of the mask variable

Points are considered as masked when the mask values is less equal than 0.

`fr.ird.osmose.grid.ECO3MGrid.java`

The `ECO3MGrid.java` class is very similar to `NcGrid.java`, excepts it has an additional parameter:

Table 2.4: Parameters for the `ECO3MGrid.java` class

<code>grid.stride</code>	Number of aggregation points
--------------------------	------------------------------

This parameter defines the number of input cells that will be aggregated together to make one osmose cell. For instance, a value of 4 implies that 16 cells of `Eco3M` grid will be used to make one cell of the Osmose model.

Note that the input values are expected to be `double`.

`fr.ird.osmose.grid.BFMGrid.java`

It is the same as `ECO3MGrid.java`, except that the input values of longitude, latitude and mask are expected to be `float`.

`fr.ird.osmose.grid.OriginalGrid`

Historically OSMOSE allows to define a regular grid, given a few parameters. Even though it is preferable to use the NetCDF grid definition, here is how a regular grid can be defined:

Table 2.5: Parameters for the EC03MGrid.java class

grid.ncolumn	Number of longitudes
grid.nline	Number of latitudes
grid.lowright.lat	Lower right latitude
grid.lowright.lon	Lower right longitude
grid.upleft.lat	Upper left latitude
grid.upleft.lon	Upper right latitude
grid.mask.file	CSV containing the land-sea mask

The mask file is a CSV with `ncolumn` and `nline`. Land takes the value `-99`, and ocean cell are defined by a `0`. Here is an example of a CSV mask file for a 4x4 grid:

Note: The `grid.ncolumn` and `grid.nline` have been renamed to `grid.nlon` and `grid.nlat` in version 3.3.3

Table 2.6: Example of a 4x4 grid file

0	0	0	0
0	0	0	-99
0	0	-99	-99
0	0	-99	-99

2.3.2 Osmose >= 4.2.0

From Osmose 4.2.0, only the `NcGrid.java` class has been kept.

Danger: Therefore, old configurations will need pre-processing in order to run with the most recent Osmose versions

2.3.3 Osmose >= 4.3.0

In Osmose >= 4.3, NaN will be considered as land points. Therefore, resource files with filled values over land can be used as a mask file.

Furthermore, an additional optional parameter has been added, `grid.var.surf`, which provides the name of the cell surface variable (which must be defined in m^2). If not provided, cell surfaces will be reconstructed from longitude and latitude coordinates.

2.4 Biotic resources

Biotic resources such as phytoplankton and zooplankton are not explicitly modelled in Osmose but are essential to take into account as they constitute the base of the trophic chain. They are considered as an input of the model, spatially explicit and varying with time.

In this section, the way LTL biotic resources are defined is described

2.4.1 Osmose \leq 4.2.0

In the Osmose versions prior to 4.2.0, the biotic resources were defined as follows:

Table 2.7: List of parameters to define biotic resources (\leq 4.2.0)

plankton.name.plk#	Name of the plankton group.
plankton.TL.plk#	Trophic level of the plankton group.
plankton.size.min.plk#	Minimum size of the organisms in the plankton group (centimeters).
plankton.size.max.plk#	Maximum size of the organisms in the plankton group (centimeters).
plankton.accessibility2fish.plk#	Fraction of the plankton biomass that is accessible to the fish, ranging from zero to one.

The `plankton.accessibility2fish.plk#` parameter accounts for many biological processes that are not explicitly represented in Osmose and basically says that only a small fraction of the plankton in the water column is effectively available to the fish for preying upon. Plankton accessibility is generally completely unknown and, just like larval mortality, it should be estimated in the calibration process.

2.4.2 Osmose \geq 4.3.0

Since Osmose 4.3.0, plankton groups are defined using the same parameters, except that they have different names.

Table 2.8: List of parameters to define biotic resources (\geq 4.3.0)

species.name.sp#	Name of the plankton group.
species.TL.sp#	Trophic level of the plankton group.
species.size.min.sp#	Minimum size of the organisms in the plankton group (centimeters).
species.size.max.sp#	Maximum size of the organisms in the plankton group (centimeters).
species.accessibility2fish.sp#	Fraction of the plankton biomass that is accessible to the fish, ranging from zero to one.
species.type.sp#	Type of the species. Must be resource for biotic resources

An additional argument, `species.type.sp#`, must be defined and set to `resource` for biotic resources.

2.5 Background species

Background species have first been introduced in Osmose by [FOT+17]. They can be viewed as an intermediary between focal species (i.e. species of interest, whose full life cycle is simulated) and lower trophic levels (plankton for instance). They differ from lower trophic levels since they can feed on focal species and be targeted by fisheries (since version 4.3)

They are available on Osmose since version 4.1.0, although their parameterization has changed in version 4.3.0.

2.5.1 Osmose <= 4.1.0

Background parameters in Osmose version 4.1.0 are parameterized as follows:

Table 2.9: Parameters for background species.

biomass.byDt.bySize.file.bkg#	CSV file containing the background species biomass by size class and by time step.
species.trophiclevel.bkg#	Array of trophic level (one for each size class)
species.name.bkg#	Name of the background species
species.length2weight.allometric.power.bkg#	Allometric factor for weight to length conversion
species.length2weight.condition.factor.bkg#	Allometric power for weight to length conversion
predation.accessibility.stage.threshold.bkg#	Threshold for accessibility matrix
predation.encyency.critical.bkg#	Critical predation success (C_{SR})
predation.ingestion.rate.max.bkg#	I_{max} (grams of food per gram of fish and per year)
predation.predPrey.stage.threshold.bkg#	Age or size thresholds for predation/prey size ratios
predation.predPrey.sizeRatio.max.bkg#	Array of R_{max} values
predation.predPrey.sizeRatio.min.bkg#	Array of R_{min} values

The biomass provided in the CSV file will then be distributed over the domain using distribution maps (similar to the ones defined for focal species).

Table 2.10: Parameters for background species.

movement.bkgSpecies.species.map#	Name of the background species to which the map is associated.
movement.bkgSpecies.class.map#	Size class for which the map is associated.
movement.bkgSpecies.season.map#	Time steps during which the map will be used
movement.bkgSpecies.year.min.map#	Initial year when the map will be used
movement.bkgSpecies.year.max.map#	Final year when the map will be used

Warning: For background species, the size classes are **fixed**!

Accessibility matrix

When using background species, the accessibility matrix must be changed accordingly. It must always have the following form:

	Focal (pred)	Background (pred)
Focal (prey)		
Background (prey)		
LTL (prey)		

Background biomass

Background species biomass is defined from a biomass time series (one per species and per size class) and by distribution maps. The distribution maps contain defines the distribution of the background species over space. They contain float values, which are normalize, so that the integral over space equals one:

$$\sum_{k=0}^{N_{cell}-1} D_k = 1$$

Before each time step, the background species biomass is reset by multiplying the biomass time series by the map distribution factor. Because of the normalisation, the spatially integrated biomass is equal to the biomass provided in the time-series.

2.5.2 Osmose >= 4.3.0

In Osmose version >= 4.3, background species are parameterized as follows:

Table 2.11: Parameters for background species (>= 4.3.0)

Species parameters	
species.name.sp#	Name of the background species
species.type.sp#	Type of the background species. Must be background
species.length2weight.allometric.power.sp#	Allometric factor for weight to length conversion
species.length2weight.condition.factor.sp#	Allometric power for weight to length conversion
predation.encyclopedia.critical.sp#	Critical predation success (C_{SR})
predation.ingestion.rate.max.sp#	I_{max} (grams of food per gram of fish and per year)
predation.predPrey.stage.threshold.sp#	Age or size thresholds for predation/prey size ratios
predation.predPrey.sizeRatio.max.sp#	Array of R_{max} values
predation.predPrey.sizeRatio.min.sp#	Array of R_{min} values
Resource forcing parameters	
species.biomass.total.sp#	Total biomass for the given ressource (will be distributed over the whole domain)
species.file.sp#	Regular expression defining the input files. Can be a file name.
species.file.caching.sp#	Resource caching method. Must be <code>none</code> , <code>incremental</code> or <code>all</code> (default).
Background species parameters	
species.nclass.sp#	Number of size classes.
species.trophiclevel.sp#	Array of trophic level (one value for each size class)
species.age.sp#	Array of school ages (one value per each size class)
species.length.sp#	Array of school lengths (one value for each size class)
species.size.proportion.sp#	Array of size proportion (one value for each size class)

The spatio-temporal distribution of background is now defined from a NetCDF file (see [Section 2.6](#)).

The distribution among size is now controlled by the `species.nclass.sp#` and `species.size.proportion.sp#` parameters.

Warning: At this time, the distribution among the size classes is constant over the entire simulation.

2.6 Resource forcing

2.6.1 Osmose <= 4.0.0

In Osmose versions <= 4.0.0, there were several ways to read the resource forcing files, depending on the value of the `ltl.java.classname` parameter.

ECO3M

If the class name is set equal to `fr.ird.osmose.ltl.LTLForcingECO3M` or `fr.ird.osmose.ltl.LTLFastForcingECO3M`, data values were read as follows.

Table 2.12: Eco3M resource parameters

<code>ltl.nstep</code>	Number of total time steps in the LTL files
<code>plankton.conversion2tons.plk#</code>	Conversion factor in tons
<code>ltl.netcdf.var.plankton.plk#</code>	Names of the Netcdf resource names
<code>ltl.netcdf.file.t#</code>	Names of the NetCDF files to read.
<code>ltl.netcdf.var.zlevel</code>	Name of the 3D depth variable
<code>grid.stride#</code>	Number of input cells that will be aggregated in an Osmose cell.
<code>ltl.integration.depth</code>	Maximum depth where to perform vertical integration

BFM

If the class name is set equal to `fr.ird.osmose.ltl.LTLForcingBFM` or `fr.ird.osmose.ltl.LTLFastForcingBFM`, data values were read as follows.

Table 2.13: BFM resource parameters

<code>ltl.nstep</code>	Number of time steps within a file.
<code>plankton.conversion2tons.plk#</code>	Conversion factor in tons
<code>ltl.netcdf.var.plankton.plk#</code>	Names of the Netcdf resource names
<code>ltl.netcdf.file.t#</code>	Names of the NetCDF files to read.
<code>ltl.netcdf.dim.ntime</code>	Number of time steps within a file
<code>grid.netcdf.file</code>	Name of the bathymetry file
<code>ltl.netcdf.var.zlevel</code>	Name of the 1D depth variable
<code>ltl.netcdf.var.bathy</code>	Name of the bathymetry variable
<code>ltl.integration.depth</code>	Maximum depth where to perform vertical integration

ROMS/PISCES

If the class name is set equal to `fr.ird.osmose.ltl.LTLForcingRomsPisces` or `fr.ird.osmose.ltl.LTLFastForcingRomsPisces`, data values were read as follows.

Table 2.14: ROMS/PISCES resource parameters

ltl.nstep	Number of time steps within a file.
plankton.conversion2tons.plk#	Conversion factor in tons
ltl.netcdf.var.plankton.plk#	Names of the Netcdf resource names
ltl.netcdf.file.t#	Names of the NetCDF files to read.
ltl.netcdf.grid.file	Name of the grid file
ltl.netcdf.var.lon	Name of the longitude variable
ltl.netcdf.var.lat	Name of the latitude variable
ltl.netcdf.var.bathy	Name of the bathymetry variable
ltl.netcdf.var.csr	Name of the CSR variable
ltl.netcdf.var.hc	Name of the Hc variable
ltl.integration.depth	Maximum depth where to perform vertical integration

Fast forcing

In the above configurations, Osmose processes the file to convert them into the right format. However, there is also the possibility to use a file already in the right format using the `fr.ird.osmose.ltl.LTLFastForcing` class.

Table 2.15: Fast forcing resource parameters

ltl.nstep	Number of time steps within a file.
ltl.netcdf.file	Name of the resource NetCDF file
plankton.biomass.total.plk#	Total biomass within the domain. If not found, reads value from NetCDF
plankton.multiplier.plk#	Multiplier of input biomass (default 1, used to run sensitivity experiments).

The netcdf file must contain a 4d variable `ltl_biomass`, whose dimensions are (time, ltl, lat, lon)

Danger: The order along the ltl dimension must be consistent with the order of definition of the plankton parameters.

Danger: With the fast forcing, the `plankton.conversion2tons.plk#` parameter is not used since data must be provided in tons!

2.6.2 Osmose 4.1.0 - 4.2.0

In versions 4.1.0 and 4.2.0, only the `FastForcing` method has been kept. However, it has been slightly improved. Now the model can take one NetCDF file per resource variable, and the variable in the NetCDF must match the `plankton.name.plk#` parameter. In this way, there is no more dependency on the order.

Table 2.16: Fast forcing resource parameters

plankton.biomass.total.plk#	Total biomass within the domain. If not found, reads value from NetCDF
ltl.nstep	Number of time steps within a file.
ltl.netcdf.file.plk#	Name of the resource NetCDF file (one for each resource)
plankton.multiplier.plk#	Multiplier of input biomass (default 1, used to run sensitivity experiments).

Danger: Resource species for which `plankton.biomass.total.plk#` is used should be defined last, i.e. their index # should be greater than those of the other species.

Danger: The resource files must be provided in tons! ****The `plankton.conversion2tons.plk#` parameter is not used.**

2.6.3 Osmose >= 4.3.0

In the 4.3.0 version, the resource forcing remains similar to versions 4.1-4.2, with some changes in parameters.

Table 2.17: Fast forcing resource parameters

<code>species.biomass.total.sp#</code>	Total biomass for the given ressource (will be distributed over the whole domain)
<code>species.file.sp#</code>	Regular expression defining the input files. Can be a file name.
<code>species.file.caching.sp#</code>	Resource caching method. Must be <code>none</code> , <code>incremental</code> or <code>all</code> (default).

If the `species.biomass.total.sp#` is not found, then the value will be read from the NetCDF file.

The `species.file.sp#` parameter can now take as an input regular expressions, which will allow to loop over all the files. The regular expressions must be defined in Java mode (see for instance XXX).

The `species.file.caching.sp#` defines how the input data will be read:

- In `all` mode, all the dataset is read at the first time step and stored into memory. *Should be used for climatological forcings for instance.*
- In `incremental` mode, each time a new time-step is read from file, it is stored in memory. Previous time-steps are kept in memory.
- In `none` mode, the data is read from file at each time-step. This mode is costly in term of runtime but light in memory since only one time-step is stored.

Note: In version 4.3.0, the resource forcing parameter also applies to background species.

2.7 Osmose time-stepping

Processes called in one Osmose time-step are show in [Fig. 2.1](#).

A detailed description of the processes and their associated parameters is given below.

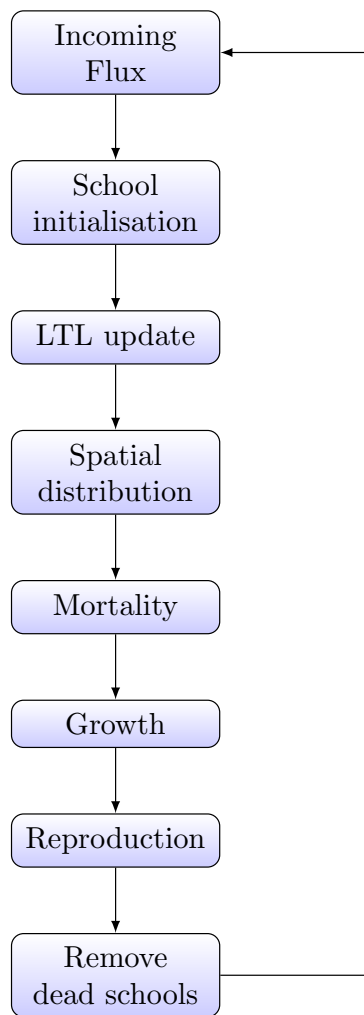


Fig. 2.1: Processes called during one Osmose time-step.

2.7.1 Input flux of biomass

Some species might not do the full life cycle within the simulated domain (reproduce outside the domain for example). For such species, one way to take them into account is to include a flux of schools with user-defined age or length at specific time of the year. This is done by setting either the `flux.incoming.byDt.byAge.file.sp#` or `flux.incoming.byDt.bySize.file.sp#` parameters, which are the paths of the CSV files containing the input flux

Table 2.18: Example of input flux by time-step and by age class

Time step / Age	0	2	3	4
0	0	500	800	0
1	0	500	800	0
2	0	400	700	0
3	0	400	700	0

The age classes (year) are automatically scanned by Osmose. In this case there are 4 classes: `[0 2[`, `[2 3[`, `[3 4[` and `[4 lifespan[`. Osmose will sets the incoming age at the middle of the interval: 1 year, 2.5 year, 3.5 year, etc. The value of the time step does not matter, Osmose assumes there is one line per time step. The number of time steps in the CSV file must be a multiple of the number of time steps per year. If the time series is shorter than the duration of the simulation, Osmose will loop over it. If the time series is longer than the duration of the simulation, Osmose will ignore the exceeding steps.

In the above example, for the first time step, Osmose will input 500 tonnes of 2.5 year old school and 800 tonnes of 3.5 year school. The incoming biomass should be calibrated. Size classes are handled the same way than age classes.

The `simulation.nschool.sp#` parameter takes a slightly different meaning for the incoming flux process. It still controls the number of schools created during the reproduction process (which may occur independently of the incoming flux process, depending on your configuration parameters) but it also controls the number of schools created for each age/size class and time step. The meanings are close enough so as not to worry about the value of this parameter and its order of magnitude depending on whether it controls reproduction, incoming flux or both.

2.7.2 School initialisation

The school initialisation process allows to reset some variables at the beginning of the time steps. For instant, the number of dead individuals, the total ingestion of the school, etc.

2.7.3 Resource update

The resource update consists in updating the resource forcings by reading the proper time-steps in the input NetCDF files. This updates is done for both resource and background species.

2.7.4 Spatial distribution

At each time-step, the spatial distribution of fished is randomly changed based either based on distribution maps or based on random walk processes.

The displacement mode is defined through the `movement.distribution.method.sp#`, whose values are either `random` or `maps`.

Random distribution

For random distribution, two parameters need to be defined:

Table 2.19: Random distribution parameters

movement.distribution.ncell.sp#	Number of cells in which the species is allowed to move
movement.randomwalk.range.sp#	Number of adjacent cells a species can use during random walk (foraging)

At the beginning of the simulation, one cell is first randomly picked up in the whole domain, and ncell are selected around it.

At each time-step, schools are moved following a random walk method within this domain, with the range defined in parameter.

Map definition

Another way to define species distribution is to use distribution maps, which can generally be obtained from niche modelling.

A single map is defined as follows:

Table 2.20: Random distribution parameters

move- ment.randomwalk.range.sp#	Number of adjacent cells a species can use during random walk (foraging)
movement.map#.file	Name of the CSV file that contains the distribution map
movement.map#.species	Name of the species associated with the map.
movement.map#.age.min	Minimum age (in years) when to use the map.
movement.map#.age.max	Maximum age (in years) when to use the map.
movement.map#.year.min	Minimum simulation time (in years) when to use the map.
movement.map#.year.max	Maximum simulation time (in years) when to use the map.
movement.map#.years	Array of years during when to use the map. (instead of setting initial and final year)
movement.map#.season	Array of time-steps during when to use the map

One map is associated to a unique species for a given age span, year span and season. The full spatial distribution of a species can be represented using as many maps as necessary to cover different age spans and/or year spans and/or seasons. Let's now have a look at each parameter in detail.

Note that the CSV file has the same number of lines and columns as the OSMOSE grid. The distribution area can be defined using either a presence/absence map (1 for presence, 0 for absence) or a map of probability of presence (containing values ranging from 0 to values <1).

The same CSV file can be used to define different maps.

If the file path is set to null it means that the schools concerned by this map are out of the simulated domain (e.g., migrating species). .. See the parameter mortality.out.rate.sp for mortality rate of species momentarily out of the simulated area. When a school comes back to the simulated area, it will be randomly located on a new map (the one corresponding to the species and age class of the school at the current time step of the simulation).

Several maps can be defined for representing the spatial distribution of a single species. For example:

```
#Map 0
movement.map0.species = euphausiids
movement.map0.file = maps/mymap_euphau1.csv
```

(continues on next page)

(continued from previous page)

```

movement.map0.age.min = 0
movement.map0.age.max = 0.2
movement.map0.year.min = 0
movement.map0.year.max = 40
movement.map0.season = 0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23

#Map 1
movement.map1.species = euphausiids
movement.map1.file = maps/mymap_euphau2.csv
movement.map1.age.min = 0.2
movement.map1.age.max = 1
movement.map1.year.min = 0
movement.map1.year.max = 40
movement.map1.season = 0;1;2;3;4;5;6;7;8;9

#Map 2
movement.map2.species = euphausiids
movement.map2.file = maps/mymap_euphau3.csv
movement.map2.age.min = 0.2
movement.map2.age.max = 1
movement.map2.year.min = 0
movement.map2.year.max = 40
movement.map2.season = 10;11;12;13;14;15;16;17;18;19;20;21;22;23

```

By increasing the number of maps, the description of the spatial distribution can be as detailed and refined as you want, as long as you have such information. It will allow for instance to create some maps for eggs (an egg in Osmose is a new school of age zero that is created during the reproduction process), some maps for the juveniles and some maps for the adults, as many as necessary to describe ontogenetic migrations.

From one time step to an other, the movement manager checks whether a given school remains in the same map or should “jump” to an other map (e.g. eggs map to juvenile map or adults in summer to adults in winter). In the latter case (change of map), the schools are relocated randomly in the new map. In the former case (same map), the movement manager mimics foraging movement with a random-walk that moves schools to immediately adjacent cells within their distribution area.

2.7.5 Growth

Individuals of a given school are assumed to grow in size and weight at a given time only when the amount of food they ingested fulfill maintenance requirements, i.e., only when their predation efficiency at that time is greater than the predation efficiency ensuring body maintenance of school.

$$G(s, a) = M_{\Delta}(s, a) \times \frac{S_R(s, a) - C_{S_R}(s)}{1 - C_{S_R}(s)} \text{ if } S_R(s, a) \geq C_{S_R}$$

$$G(s, a) = 0 \text{ if } S_R(s, a) < C_{S_R}$$

with C_{S_R} is the critical predation efficiency and S_R the predation success rate of the school.

$M_{\Delta}(s, a) = \lambda \times \Delta L(a)$ is the maximum growth rate at age a and for species s .

$\Delta L(a) = L(a+1) - L(a)$ is the mean length increase determined from a growth function (Von Bertalanffy or Gompertz growth function), while λ is a factor that allows to control the maximum length at a given age.

Table 2.21: Growth parameters

growth.java.classname.sp#	Class name of the age to length conversion
predation.efficiency.critical.sp#	Critical predation success (C_{SR})
species.delta.lmax.factor.sp#	λ (default = 2)

Von Bertalanffy growth

When the `growth.java.classname.sp#` is equal to `fr.ird.osmose.process.growth.VonBertalanffyGrowth.java`, a von Bertalanffy growth function is used. **It is the default one.**

$$\begin{aligned}
 L(a) &= L_{egg} \text{ if } a = 0 \\
 L(a) &= L_{egg} + (L_{thres} - L_{egg}) \times \left(\frac{a}{a_{thres}} \right) \text{ if } a > 0 \text{ \& } a < a_{thres} \\
 L(a) &= L_{\infty} \times \left(1 - \exp^{-K(age-t_0)} \right) \text{ else}
 \end{aligned}$$

with

$$L_{thres} = \min \left[L_{egg}, L_{\infty} \times \left(1 - \exp^{-K(a_{thres}-t_0)} \right) \right]$$

A Von Bertalanffy model is used to calculate mean length increase above a threshold age a_{thres} determined for each HTL group from the literature. Below a_{thres} , a simple linear model is used. The rationale behind this is that Von Bertalanffy parameters are usually estimated from data excluding youngs of the year or including only very few of them. Assuming a linear growth between age 0 and a_{thres} ensures a more realistic calculation of mean length increases for early ages of HTL groups ([TSJ+09]).

Table 2.22: Von Bertalanffy parameters

species.linf.sp#	L_{inf} (cm)
species.k.sp#	K
species.t0.sp#	t_0
species.vonbertalanffy.threshold.age.sp#	a_{thres} (years, default=1 year)

Gompertz growth

When the `growth.java.classname.sp#` is equal to `fr.ird.osmose.process.growth.GompertzGrowth.java`, a Gompertz growth function is used.

$$\begin{aligned}
 L(a) &= L_{egg} \text{ if } a = 0 \\
 L(a) &= L_{start} \times \exp^{K_e \times a} \text{ if } a > 0 \text{ \& } a < a_{exp} \\
 L(a) &= L_{exp} + (L_{gom} - L_{exp}) \frac{a - a_{exp}}{a_{gom} - a_{exp}} \text{ if } a > a_{exp} \text{ \& } a < a_{gom} \\
 L(a) &= L_{inf} \times \exp^{-\exp^{-K_g(a-t_g)}} \text{ else}
 \end{aligned}$$

with

$$\begin{aligned}
 L_{exp} &= L_{start} \times \exp^{K_e \times a_{exp}} \\
 L_{gom} &= L_{inf} \times \exp^{-\exp^{-K_g(a_{gom}-t_g)}}
 \end{aligned}$$

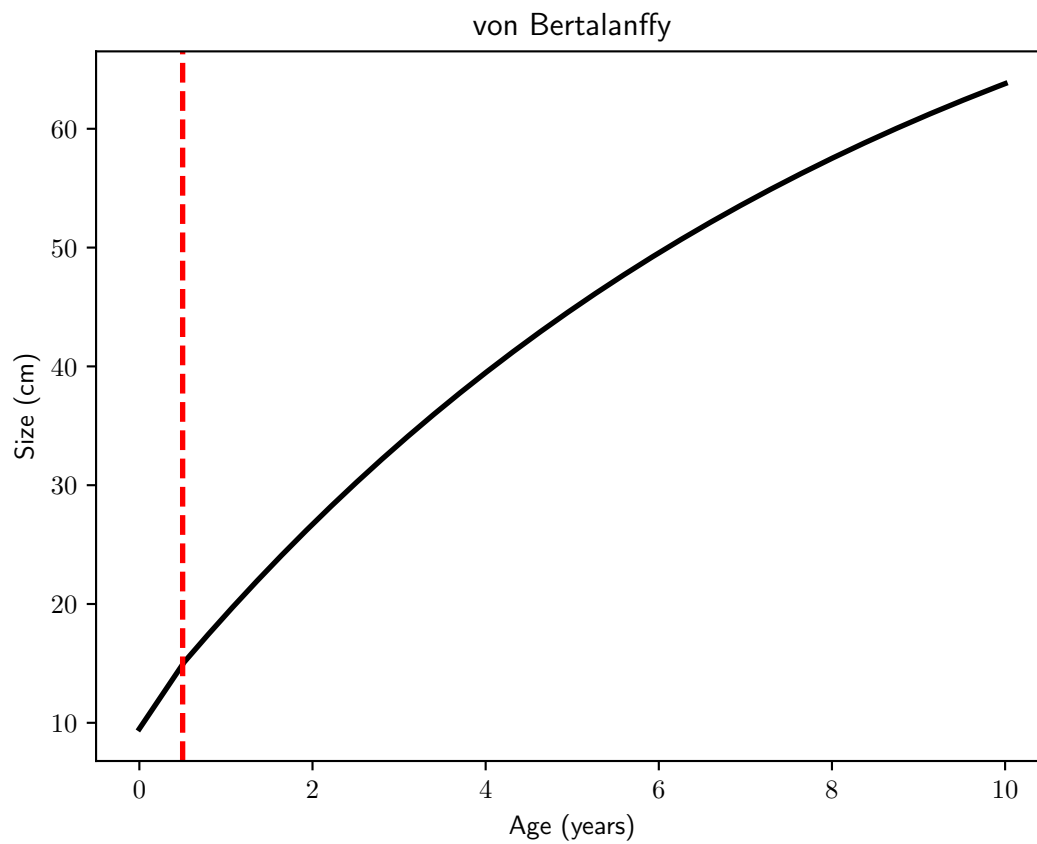


Fig. 2.2: Von Bertalanffy growth curve

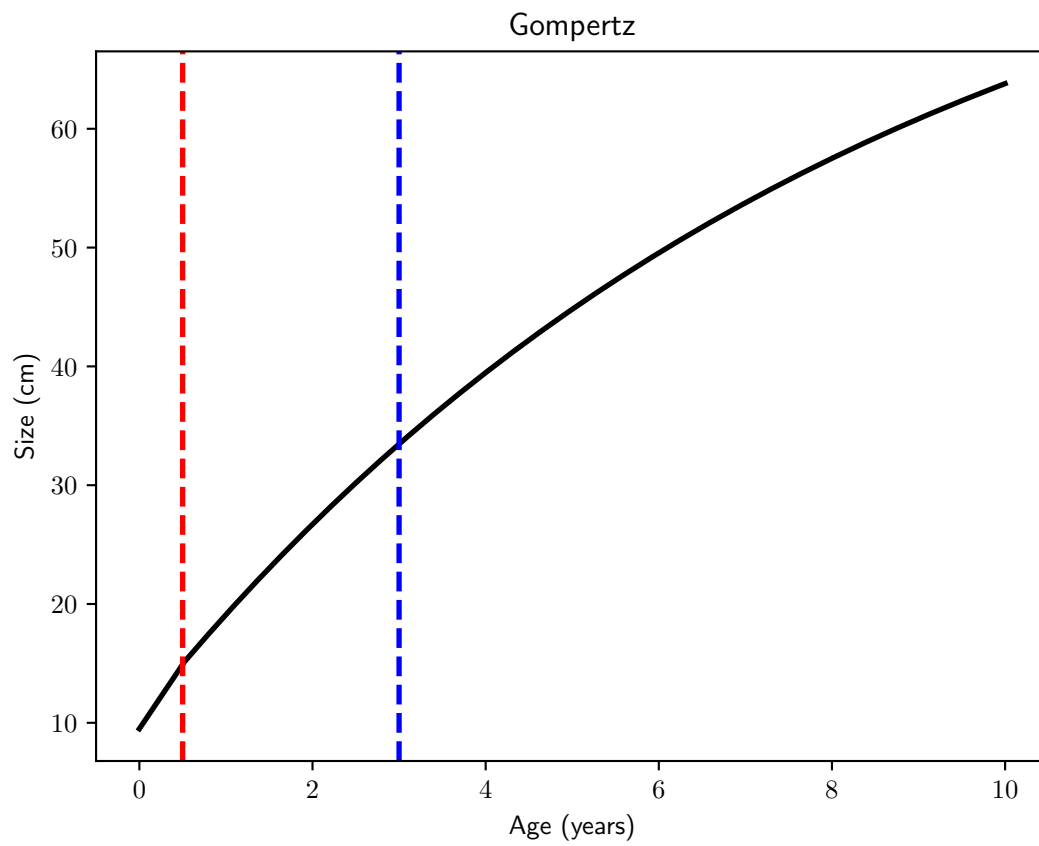


Fig. 2.3: Gompertz growth curve

Table 2.23: Gompertz parameters

growth.exponential.lstart.sp#	L_{start} (cm)
growth.exponential.ke.sp#	K_e
growth.gompertz.linf.sp#	L_{inf} (cm)
growth.gompertz.kg.sp#	K_g
growth.gompertz.tg.sp#	t_g (years)
growth.exponential.thr.age.sp#	a_{exp} (years)
growth.gompertz.thr.age.sp#	a_{gom} (years)

2.7.6 Reproduction

The system starts from a pristine state, with no schools in the domain. For a few years (user-defined), Osmose will release some eggs for every species. The eggs enter the different steps of the life cycle, and once the fish reach sexual maturity, the reproduction process takes over. Osmose stops the seeding, unless the spawning stock biomass gets depleted. In that case Osmose resumes the seeding by releasing some eggs until there are again mature individuals in the system for carrying on the reproduction process. Osmose completely ceases the seeding when the simulation reaches the maximal number of years for seeding (user-defined).

For a given species:

$$B_{mat} = \sum_{maturesch.} B$$

$$N_{eggs} = FRAC_{fem} * \alpha * season * B_{mat} \text{ if } SSB > 0$$

$$N_{eggs} = FRAC_{fem} * \alpha * season * B_{seeding} \text{ if } SSB = 0$$

By following this approach:

- No assumption is made about the structure of the populations but it emerges from individual interactions
- it reduces computing requirements for the spin-up as the first years are the fastest to run
- it reduces the number of time steps for the spin-up
- it minimizes the amplitude of population oscillations

Finally, the seeding biomass is then used to add new schools to the system, depending on the value of N_{eggs} .

If $N_{eggs} < N_s$:

$$N_{sch} = 1$$

$$A_{sch} = N_{eggs}$$

else:

$$N_{sch} = N_s$$

$$A_{sch} = \frac{N_{eggs}}{N_s}$$

Table 2.24: Reproduction paramters

simulation.nschool.sp#	Number of schools of species # to create during reproduction (N_s)
simulation.nschool	Number of schools to create during reproduction (N_s). Used if no species specific value provided
population.seeding.biomass.sp#	Seeding biomass ($B_{seeding}$, tons)
reproduction.season.file.sp#	File providing the seeding distribution within a year
species.sexratio.sp#	Fraction of females ($FRAC_{fem}$)
species.relativefecundity.sp#	Number of eggs per gram of mature female (relative fecundity)
population.seeding.year.max	Number of years when the artificial seeding is activated

2.8 Mortality

Within each time step, the total mortality of a given school is comprised of predation mortality caused by other schools, starvation mortality, fishing mortality, and diverse other natural mortality rate. The four different mortalities are computed so as to represent quasi simultaneous processes, and we consider that there is competition and stochasticity in the predation process.

Within each time step, OSMOSE considers each pair of school/source of mortality in turn in a random order. To ensure that the random order of the mortality sources and of the schools does not bias the resulting instantaneous mortality rates applied and effectively correspond to the mortality rates specified in input (for fishing and diverse natural mortality), all the mortality events are iterated within a time step over a fixed number of sub-time step.

Table 2.25: Mortality parameters

stochastic.mortality.seed	Integer to fix the random number generator.
mortality.subdt	Number of mortality sub time-steps.
mortality.algorithm	Mortality algorithm (iterative or stochastic) (Osmose <= 4.2.0)

Note: In Osmose >= 4.3.0, the `mortality.algorithm` parameter is no more used. Stochastic mortality algorithm is used.

Mortality processes are detailed below.

2.8.1 Predation mortality

The central assumption in OSMOSE is that predation is an opportunistic process, which depends on:

- the overlap between predators and potential prey items in the horizontal dimension
- size adequacy between the predators and the potential prey (determined by **predator/prey size ratios**); and when the information is available
- the accessibility of prey items to predators, which depends on their vertical distribution (this being determined by means of **accessibility coefficients**). Thus, in OSMOSE, the food web structure emerges from local predation and competition interactions.

Consider a predator school S_{pred} .

Size predation

Size-predation matrix is controlled by two parameters. The predator school S_{pred} can only feed on prey schools whose length meets:

$$R_{min} \leq \frac{L_{pred}}{L_{prey}} \leq R_{max}$$

with R_{min} and R_{max} the maximum and minimum predator/prey size ratios. Therefore, the minimum and maximum sizes of a prey that a predator can eat is given by:

$$L_{max} = \frac{L_{pred}}{R_{max}}$$

$$L_{min} = \frac{L_{pred}}{R_{min}}$$

Table 2.26: Size-predation parameters

predation.predPrey.stage.structure	Structure to determine thresholds for predator/prey size ratios (age or size)
predation.predPrey.stage.threshold.sp#	Array of age or size thresholds
predation.predPrey.sizeRatio.max.sp#	Array of R_{max} values
predation.predPrey.sizeRatio.min.sp#	Array of R_{min} values

Danger: $R_{min} > R_{max}$

Since resource groups are defined by a range of sizes, and not by a single sizes, the predator will feed on a given percentage of the resource:

$$R_{rsc} = \frac{\min(L_{max_{rsc}}, L_{max}) - \max(L_{min_{rsc}}, L_{min})}{L_{max_{rsc}} - L_{min_{rsc}}}$$

which is the overlapping range of the predator accessible range and of the resource size range.

Accessibility

First, the accessibility of all the preys to school S_{pred} is determined from an accessibility matrix for every species and stages. This matrix must not be used to define diet preferences but rather to take into account for a difference of positions in the water column (meaning some schools might evolve around the same geographical area but never meet because they do not occur at the same depth).

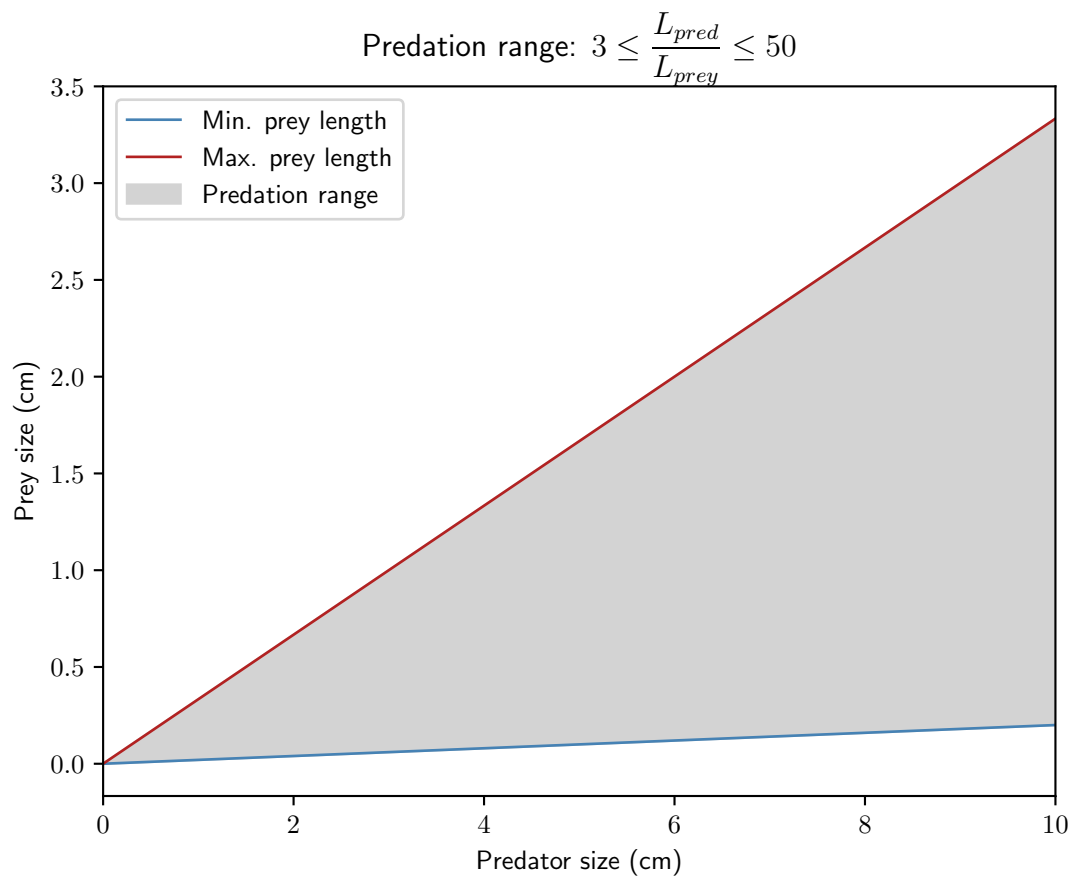


Fig. 2.4: Size-based predation in Osmose

Table 2.27: Example of a CSV predation accessibility file.

Prey / Predator	lesserSpotted < 0.45	lesserSpotted	redMullet < 0.25	redMullet
lesserSpottedDogfish < 0.45	0.05	0	0	0.05
lesserSpottedDogfish	0	0.8	0.4	0
redMullet < 0.25	0	0.4	0.8	0
redMullet	0.8	0.4	0	0.8
pouting < 0.25	0	0.4	0.8	0
pouting	0	0.8	0.4	0
whiting < 0.25	0	0.4	0.8	0
whiting	0	0.8	0.4	0
Dinoflagellates	0	0.5	1	0
Diatoms	0	0.5	1	0
Microzoo	0	0.5	1	0
Mesozoo	0	0.5	1	0
Macrozoo	0	0.5	1	0
VSBVerySmallBenthos	1	0.5	0	1
SmallBenthos	1	0.5	0	1
MediumBenthos	1	0.5	0	1
LargeBenthos	1	0.5	0	1
VLBVeryLargeBenthos	1	0.5	0	1
backgroundSpecies	0	0	0	0

Each line of the matrix corresponds to a prey (including plankton groups), and each column to a predator. The file must be understood as follow: lesserSpottedDogfish of age class less than 0.45 (line 1) are only accessible to young lesserSpottedDogfish (5%) and old redMullet (5%).

Table 2.28: Parameters for accessibility

predation.accessibility.stage.structure	Threshold type. Must be age or size.
predation.accessibility.stage.threshold.sp#	Array containing the stage thresholds for a given species.
predation.accessibility.file	CSV file containing the accessibility matrix

Warning: In versions <= 4.2.0, the order of the rows and columns must follow the indexing of species (focal, background and resource) and stages (e.g., species0; species1; species2 stage0; species2 stage1; species3). The threshold values provided in the CSV file are not used in this version

Since version 4.3.0, the `predation.accessibility.stage.threshold.sp#` parameter has been deprecated, since the thresholds are read directly from the CSV files by matching the < character. It is assumed that if there is no match, no threshold is provided. However, when < is matched, it is assumed that what follows is the upper bound of the class.

Furthermore, the column and row order is no more important, since a match of the species name is performed.

Finally, since version 4.3.0, accessibility matrix can vary over time with the following parameters, which follow the parameterization of movements.

Table 2.29: Parameters for time varying accessibility

predation.accessibility.file.acc#	CSV file containing the accessibility matrix
predation.accessibility.initialYear.acc#	Start year when to use the accessibility matrix
predation.accessibility.finalYear.acc#	Start year when to use the accessibility matrix
predation.accessibility.years.acc#	List of years when to use the map (instead of setting initial and final years)
predation.accessibility.steps.acc#	List of time steps when to use the map

Danger: If the `predation.accessibility.file` (with no `.acc` suffix) is found, Osmose will assume constant predation accessibility matrix.

Predation rate

Finally, the predation rate is computed as follows. First, the total accessible biomass for the predator school is computed:

$$P_{tot} = \sum_{p=preys} A(pred, prey) \times B_{prey}$$

The total biomass that a predator can eat is also computed as follow:

$$P_{eatable} = \frac{B_{pred} \times I_{max}}{N_{mort}}$$

with `N_mort` the number of sub-step of mortality processes, `B_pred` the total biomass of predator and `I_max` the maximum ingestion rate for each species, expressed in grams of food per gram of fish and per year. It is assumed that predator eat as much as they can.

The effective biomass that will be eaten by the predator is

$$P_{eaten} = \min(P_{tot}, P_{eatable})$$

Finally, for each prey, the biomass eaten by the predator is given by:

$$P_{lost} = P_{eaten} \times \frac{A(pred, prey) \times B_{prey}}{P_{tot}}$$

Finally, the success rate is computed as:

$$S_R = \frac{P_{eaten}}{P_{eatable}}$$

Table 2.30: Ingestion parameter

predation.ingestion.rate.max.sp#	I_{max} (grams of food per gram of fish and per year)
----------------------------------	---

2.8.2 Starvation mortality

Starvation mortality is applied as follows:

$$N_{starv} = N \times (1 - e^{-M_{starv}})$$

with M_{starv} the starvation mortality rate, which is computed as follows:

$$M_{starv} = M_{max} \times \left(1 - \frac{S_R}{C_{S_R}}\right) \text{ if } S_R \leq C_{S_R}$$

Note: Starvation mortality rate applied at time step t is based on the predation success computed at time-step $t - 1$

Table 2.31: Starvation mortality parameters

mortality.starvation.rate.max.sp#	Maximum rate of starvation mortality M_{max}
predation.efficiency.critical.sp#	Critical predation success rate C_{S_R}

2.8.3 Migration mortality

When a school is out of the domain, none of the standards processes (predation, growth, fishing, natural mortality, growth, starvation) apply.

The migration mortality is used to simulate all sources of mortality outside the simulated area. It applies as long as the school is located out of the simulated area

$$N_{out} = N \times (1 - e^{-M_{out}})$$

Table 2.32: Parameters for migration mortality

mortality.out.rate.sp#	Annual mortality rate for species that move out of the simulated area (M_{out})
------------------------	---

2.8.4 Fishing parameters (Osmose <= 4.0.0)

On Osmose versions **previous** to 4.0.0, fishing mortality was species-specific. It was parameterized either by providing fishing rates or catches.

Table 2.33: Fishing parameters (< 4.0.0)

mortality.fishing.type	Whether fishing mortality is provided as rate or catches
mortality.fishing.recruitment.age.sp#	Age at which a species can be fished (years)
mortality.fishing.recruitment.size.sp#	Size at which a species can be fished (cm)

Warning: Osmose does not accept fishing mortality rates for some species and catches for other species.

By rate

If mortality rate is provided, the number of dead fishes in a school is computed as follows.

$$N_{fishing} = N \times (1 - \exp^{-F})$$

Osmose offers several degrees of refinement for inputting the fishing mortality: constant, seasonal, interannual and interannual with age or size class. Depending on the available information for each species of the configuration, one must choose the best way to input it. Each species can be parameterized independently from an other. For instance

fishing mortality for species zero is a constant annual rate and fishing mortality for species three is provided as a time series per size class.

Table 2.34: Parameters for fishing rate.

mortality.fishing.rate.byDt.byAge.file.sp#	CSV file containing the fishing rates by age and by time-step
mortality.fishing.rate.byDt.bySize.file.sp#	CSV file containing the fishing rates by size and by time-step
mortality.fishing.rate.byYear.file.sp#	File containing the fishing rates by year
mortality.fishing.rate.sp#	Annual fishing rate
mortality.fishing.season.distrib.file.sp#	File containing the seasonal distribution of fishing mortality. If not provided, assumes constant fishing rate

Osmose will first look for any of the first two parameters. If not found, it will look for the third one. If not found, Osmose will finally look for the fourth one. If the third or fourth parameter are found, it will also look for the fifth one.

By catches

If mortality is provided by catches, the number dead individuals is computed as follows:

$$N_{fish} = \min \left(N, C \times \frac{B_{fish}}{B_{fishable} \times W} \right) \text{ if } B_{fishable} > 0$$

with C the catches, B_{fish} the fish biomass, W its weight and $B_{fishable}$ the biomass that can be fished.

Table 2.35: Parameters for fishing catches.

mortality.fishing.catches.byDt.byAge.file.sp#	CSV file containing the fishing rates by age and by time-step
mortality.fishing.catches.byDt.bySize.file.sp#	CSV file containing the fishing rates by size and by time-step
mortality.fishing.catches.byYear.file.sp#	File containing the fishing rates by year
mortality.fishing.catches.sp#	Annual fishing rate
mortality.fishing.season.distrib.file.sp#	File containing the seasonal distribution of fishing mortality

Note: Catches are assumed to be in tons.

Osmose will first look for any of the first two parameters. If not found, it will look for the third one. If not found, Osmose will finally look for the fourth one. If the third or fourth parameter are found, it will also look for the fifth one.

Marine Protected Areas (MPAs)

The user can defined as many MPA as he wishes.

Table 2.36: Parameters for setting MPA

mpa.file.mpa#	File containing the MPA definition
mpa.start.year.mpa#	First year when this MPA is active
mpa.end.year.mpa#	Last year when this MPA is active

The map is a CSV file similar to the movement maps. The CSV file has the same number of lines and columns as the OSMOSE grid. The MPA file must contain 1 where the MPA is defined, 0 elsewhere.

Start year and end year parameters define the time span when the MPA is enabled.

The MPA are handled within the fishing process. Every time there is a new MPA to be activated or deactivated, Osmose updates the correction factor that will be applied to the fishing mortality rates in order to take into account the uniform redistribution of the fishing effort outside the MPAs.

2.8.5 Fishing parameters (Osmose >= 4.2.0 & <= 4.2.0)

Between the 4.0.0 and 4.2.0 Osmose versions, a new way to implement fisheries is possible.

New algorithm

A new feature provided by the new fisheries implementation is the consideration of bycatch.

The `fisheries.catch.matrix.file` parameter provides the location of a CSV file containing $N_{fisheries}$ lines and $N_{species}$ columns.

The table provides, for each fisheries, the proportion of each fish biomass that is extracted by the fisherie.

In the example provided in [Fig. 2.5](#), fisherie 7 targets the sole (proportion of 100%) but with 10% of bycatches of plaice and horse mackerel.

Let's call this proportion matrix $A_{N_{fisherie}, N_{specie}}$.

The new fishing mortality process, for a given specie s , is done by computing the mortality rate associated with each fisherie:

$$N_{x,y,s,t-1} = N_{x,y,s,t} \times \exp \left(- \sum_{k=1}^{N_{fisheries}} A_{k,s} F_{k,s,t,x,y} \right)$$

with $N_{s,t}$ the number of individuals of species s and $F_{s,k}$ the fishing mortality rate.

Note: If the stochastic mortality algorithm is used, the order in which the fisheries are accessed is random.

In the new implementation of fisheries, the fishing mortality rate $F_{s,k}$ is variable in space, time and depends on the size of the targetted specie:

$$F_{k,s,t,x,y} = R_k(t) \times S_k(s) \times Z_k(x, y, t) \quad (2.1)$$

with R_k the time varying fishing rate, S_k the size-dependent multiplication factor and Z_k the space-dependent multiplication factor, whose calculation is detailed below.

Size/Age selectivity

In order to take into account the selectivity of the fisheries on the size or age of the focus species, fishing rate are now scaled by a selectivity curve ($S_k(s)$ factor in equation (2.1)).

First, the user has to define, for each fisherie, which variable will be used to compute the selectivity (age or len). This is done by filling the `fisheries.select.var.fisX` parameter.

Then, the user needs to define which kind of selectivity curve should be used. This is done by setting the `fisheries.select.curve.fisX` parameter, which can take three values.

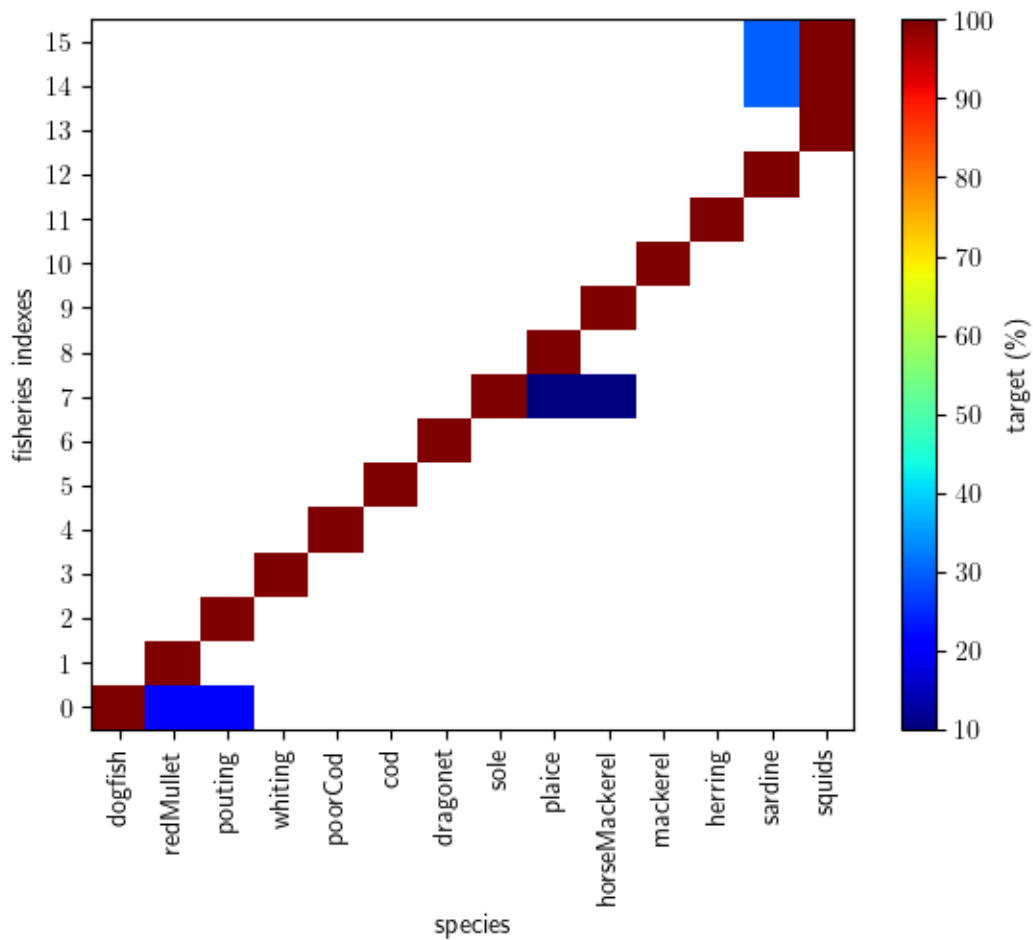


Fig. 2.5: By catch matrix.

Step selectivity

If `fisheries.select.curve.fisX = step`, a step like selectivity is used:

$$S(x) = \begin{cases} 0, & \text{if } x < x_0. \\ 1, & \text{otherwise.} \end{cases}$$

with x_0 being the `fisheries.select.l50.fisX` parameter

Sigmoid selectivity

If `fisheries.select.curve.fisX = sigmo`, a sigmoid like selectivity is used:

$$S(x) = \frac{1}{1 + \alpha \exp^{-\beta(x-x_0)}}$$

with x_0 being the `fisheries.select.l50.fisX` parameter, α the `fisheries.select.a.fisX` parameter and β the `fisheries.select.b.fisX` parameter.

Gaussian selectivity

If `fisheries.select.curve.fisX = gauss`, a Gaussian like selectivity is used:

$$S(x) = \exp^{-\gamma(x-x_0)^2}$$

with x_0 being the `fisheries.select.l50.fisX` parameter and γ the `fisheries.select.b.fisX` parameter.

Examples

The three selectivities are shown in figure [Fig. 2.6](#) with $x_0 = 100$, $\alpha = 1$, $\beta = 0.1$ and $\gamma = 0.0005$.

Time-variability

The time varying fishing rates $R_k(t)$ (equation (2.1)) can be provided using different methods, which are displayed in [Fig. 2.7](#) (for a 24 time step simulation of 2 years).

The method is set through the `fisheries.rate.method.fisX` parameter, which can take 5 values.

Constant

If `fisheries.rate.method.fisX = constant`, a constant fishing rate (`fisheries.rate.const.rate.fisX` parameter) is applied during the simulation. The model assumes that this fishing rate is a yearly rate and is converted into a seasonal one (i.e. it is divided by the `ndtPerYear` variable).

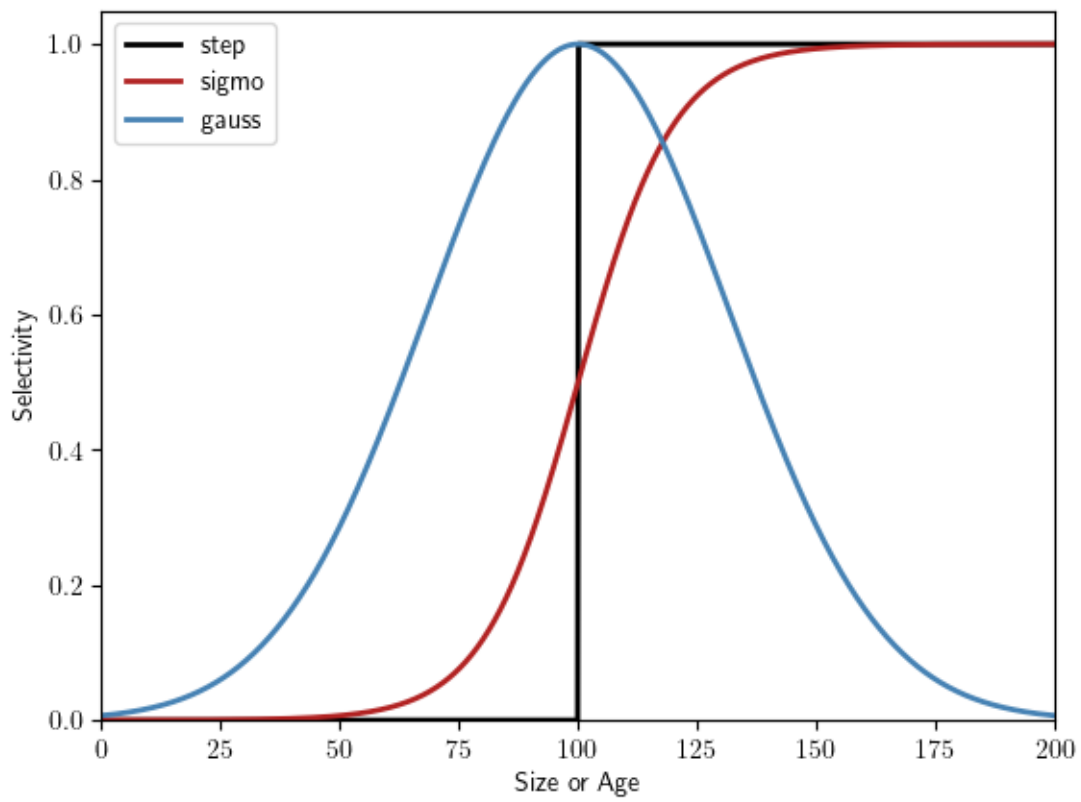


Fig. 2.6: Examples of selectivity curves.

By year

If `fisheries.rate.method.fisX = byyear`, the user provides the number of different fishing rates per year (`fisheries.rate.periodspereyear.fisX` parameter) and the associated annual fishing rates (`fisheries.rate.byyear.rate.fisX` parameter). This fishing rate will be repeated for each simulated year.

In the example below (top right), 4 fishing rates per year are provided. Therefore, the fishing rate is updated every $12 / 4 = 3$ time steps (cf. red lines).

Linear

If `fisheries.rate.method.fisX = linear`, the user provides an initial fishing rate (`fisheries.rate.linear.rate.fisX` parameter), a slope (`fisheries.rate.linear.slope.fisX`) and the number of different annual fishing rates per year (`fisheries.rate.periodspereyear.fisX` parameter). The fishing rates are computed by using an affine function.

In the example below (middle right), 4 fishing rates per year are provided. Therefore, the fishing rate is updated every $12 / 4 = 3$ time steps (cf. red lines).

Regime shifts

If `fisheries.rate.method.fisX = byregime`, the user defines time-steps of regime shifts (`fisheries.rate.regime.shifts.fisX` parameter) and their associated annual fishing rates (`fisheries.rate.regime.rates.fisX` parameter) associated with the regimes. Note that `len(shifts) + 1 = len(rates)`.

In the example below (bottom left), there are 4 regimes: at time-steps [0, 2], [3-4], [5-14], [15-23] (cf. purple lines).

By Dt

If `fisheries.rate.method.fisX = bydt`, the user provides a fishing rate for each simulation time-step (`fisheries.rate.bydt.rate.fisX`). However, if the number of provided rates is less than the total number of simulation time-steps, the fishing rate is repeated cyclically.

In the example below (middle left), only 5 rates are provided. These rates will be applied for steps [0-4], [5-9], etc. (cf. blue lines)

Warning: It is the only case where the fishing rate is assumed to be seasonal. In all other cases, it is assumed that annual rates are provided

Spatial variability

Another feature that is implemented in the new fisheries implementation is the possibility to define spatial variability in the fishing effort ($Z_k(x, y, t)$ factor in equation (2.1)).

The spatial variability is managed in the same way as map fishing movements.

A fishing map must be associated with a fisherie index. This is done by setting the `fisheries.fishmap.find.fmapX` parameter, which have values ranging from 0 to $N_{fisheries} - 1$.

The user can also provide the years at which the given map is applied (`fisheries.fishmap.year.min.fmapX` and `fisheries.fishmap.year.max.fmapX` parameters). By default, the start and end simulation years are used.

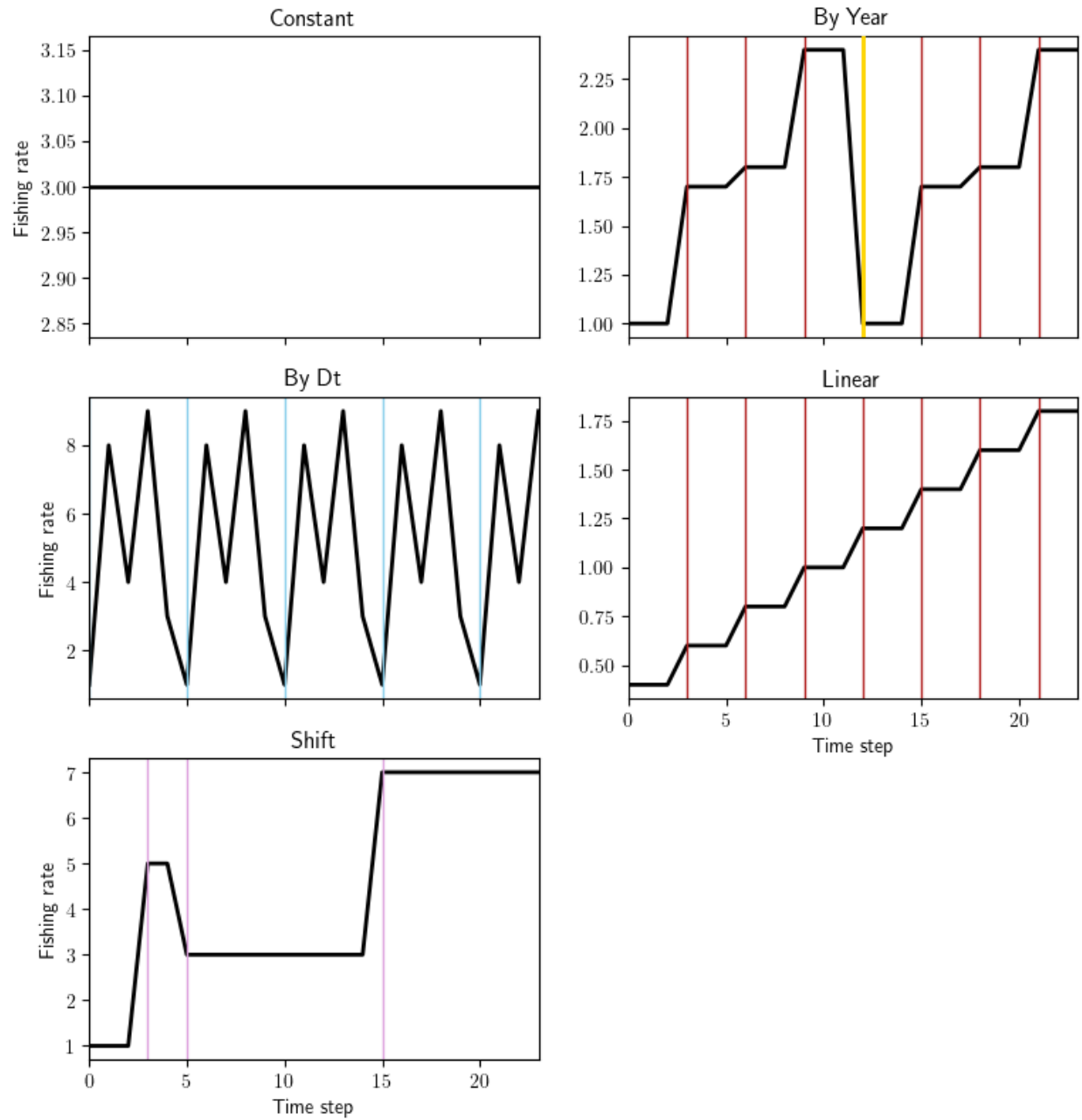


Fig. 2.7: Examples of time varying fishing rates.

The user must also provide the season at which the fishing map is used (`fisheries.fishmap.season.fmapX` parameters).

Note: If a map is missing (at a given year or during a given season), it is considered that the fisherie is deactivated.

Finally, the user must provide the CSV file containing the spatial factors (`fisheries.fishmap.file.fmapX`).

Note: The model automatically normalize the fishing efforts so that their surface weighted means is 1.

Let's call M the weighted mean of the provided factors:

$$M = \frac{\sum_{k=1}^{N_{cell}} S_k f_k}{\sum_{k=1}^{N_{cell}} S_k}$$

with N_{cell} the number of cells within the fisherie area, S_k the cell surface and f_k the provided spatial fishing factor.

The new cell factors will be $f_{k_{new}} = \frac{f_k}{M}$

For instance, if a fishing area contains 6 cells, with spatial factors of $[1, 1, 1, 2, 2]$, the new factors will be $[0.71, 0.71, 0.71, 1.43]$.

If the factors are $[1, 1, 1, 0, 0]$ (0 if a MPA overlaps with the fishing area for instance), the new factors will become $[1.67, 1.67, 1.67, 0, 0]$, which can be viewed as a uniform redistribution of the fishing effort.

2.8.6 Fishing parameters (Osmose >= 4.3)

Fishing mortality rates

In the Osmose versions >= 4.3, changes in the parameterization of fisheries have been implemented, although the spirit remains close to the one in versions 4.

Fisheries mortality time-series for each gear are now the product of three components:

- A vector of fishing mortality rates associated with fishing periods, F_{period}
- A vector of seasonality values, which provides the time-variation of the fishing effort during a given season, F_{season}
- A vector of multipliers, which provides a multiplication factor, F_{base} .

Therefore, for a given time-step t , the value of the fishing mortality for a given fleet would be:

$$F(t) = F_{period}(t) \times F_{season}(t) \times F_{base}(t)$$

Since the parameterisation is a bit tricky, a parameter (`fisheries.check.enabled`) allows to control whether the values of F , F_{period} , F_{season} and F_{base} should be saved. It allows to control that the given time-series are as expected.

Fishing base (F_{base})

The fishing base mortality multiplier is provided as regime shifts.

Table 2.37: Fishing base parameters

fisheries.rate.base.fsh#	Fishing base for the different regimes.
fisheries.rate.base.shift.fsh#	Years of the regime shifts.
fisheries.rate.base.log.enabled.fsh#	True if fisheries are provided in logscale.

If one value is provided in `fisheries.rate.base.fsh#`, this value will be used during all the simulation. If N values are provided, then the `fisheries.rate.base.shift.fsh#` parameter must contain $N - 1$ values, which are the years of the regime shifts.

Fishing period (F_{period})

There is now the possibility to define a fishing period, i.e. a period when the fishery is active.

Table 2.38: Fishing period parameters

fisheries.period.number.fsh#	Number of fishing periods within one year (N_{per})
fisheries.period.start.fsh#	Start of the active fishing period (fraction of year, default = 0, Y_{start})
fisheries.rate.byperiod.fsh#	Fishing mortality rate. Must me in $year^{-1}$

Note that the number of values expected in the `fisheries.rate.byperiod.fsh#` parameter depends on Y_{start} .

If $Y_{start} = 0$, then $N_{per} \times N_{year}$ values are expected (one value for each fishing and non-fishing season).

If $Y_{start} \neq 0$, then $N_{per} \times N_{year} + 1$ values are expected.

Fishing seasonality

In order to distribute the fishery mortality over the season, the user can define a seasonality vector, either as a file, or as a vector.

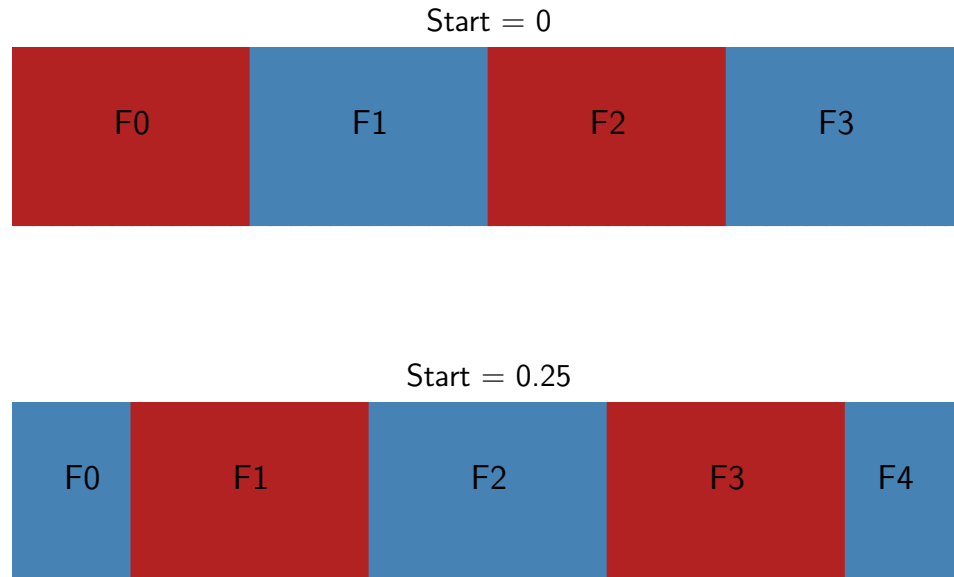
Table 2.39: Fishing seasonality parameters

fisheries.seasonality.fsh#	Array of fishing seasonality. Must contain $\frac{N_{step/year}}{N_{season}}$
fisheries.seasonality.file.fsh#	File containing the fishing seasonalities (must contain N_{step} values)

In the first case, the same seasonality will be applied for each fishing season. Imagine that we have 24 time-steps per year and two fishing season (with no offset, top of figure Fig. 2.8), then the seasonality provided should contain 12 values, which would apply for the active fishing period (green zone).

In the latter case, it is up to the user to generate the proper time series and to store it in a file.

Danger: The sum of fishing seasonalities must equal one over the fishing seasons! **No automatic normalisation is performed by Osmose!**

Fig. 2.8: Fishing period for two values of Y_{start}

Case studies

```

fisheries.rate.base.fsh0;1
fisheries.season.number.fsh0;1
fisheries.rate.byperiod.fsh0;1
fisheries.season.start.fsh0;0
fisheries.seasonality.fsh0;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.
↪04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;
↪0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166;0.04166

```

```

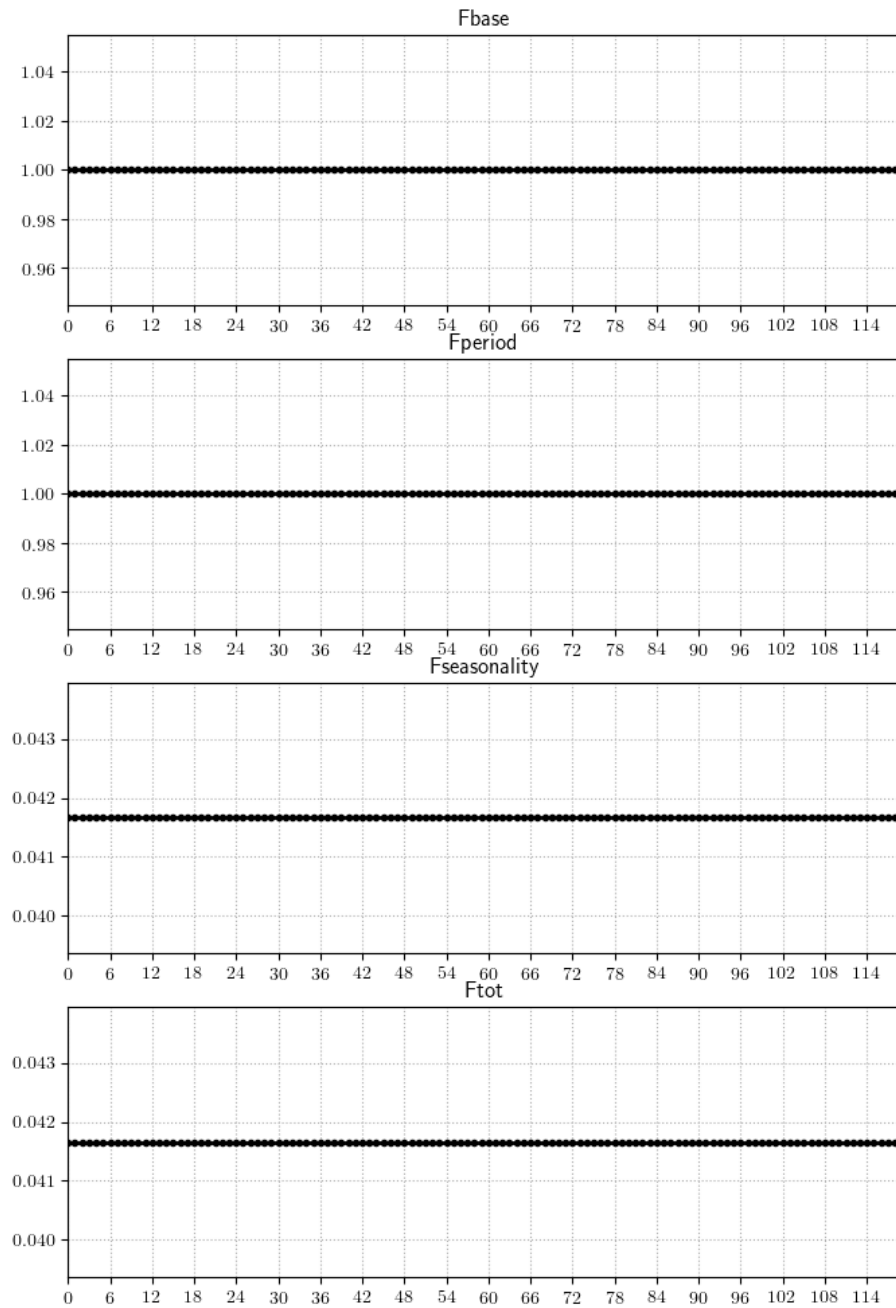
fisheries.rate.base.fsh0;1
fisheries.season.number.fsh0;2
fisheries.rate.byperiod.fsh0;1, 0, 2, 0, 3, 0, 4, 0, 5, 0
fisheries.season.start.fsh0;0
fisheries.seasonality.fsh0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.1666;0.1666;0.1666;0.1666;0.
↪1666;

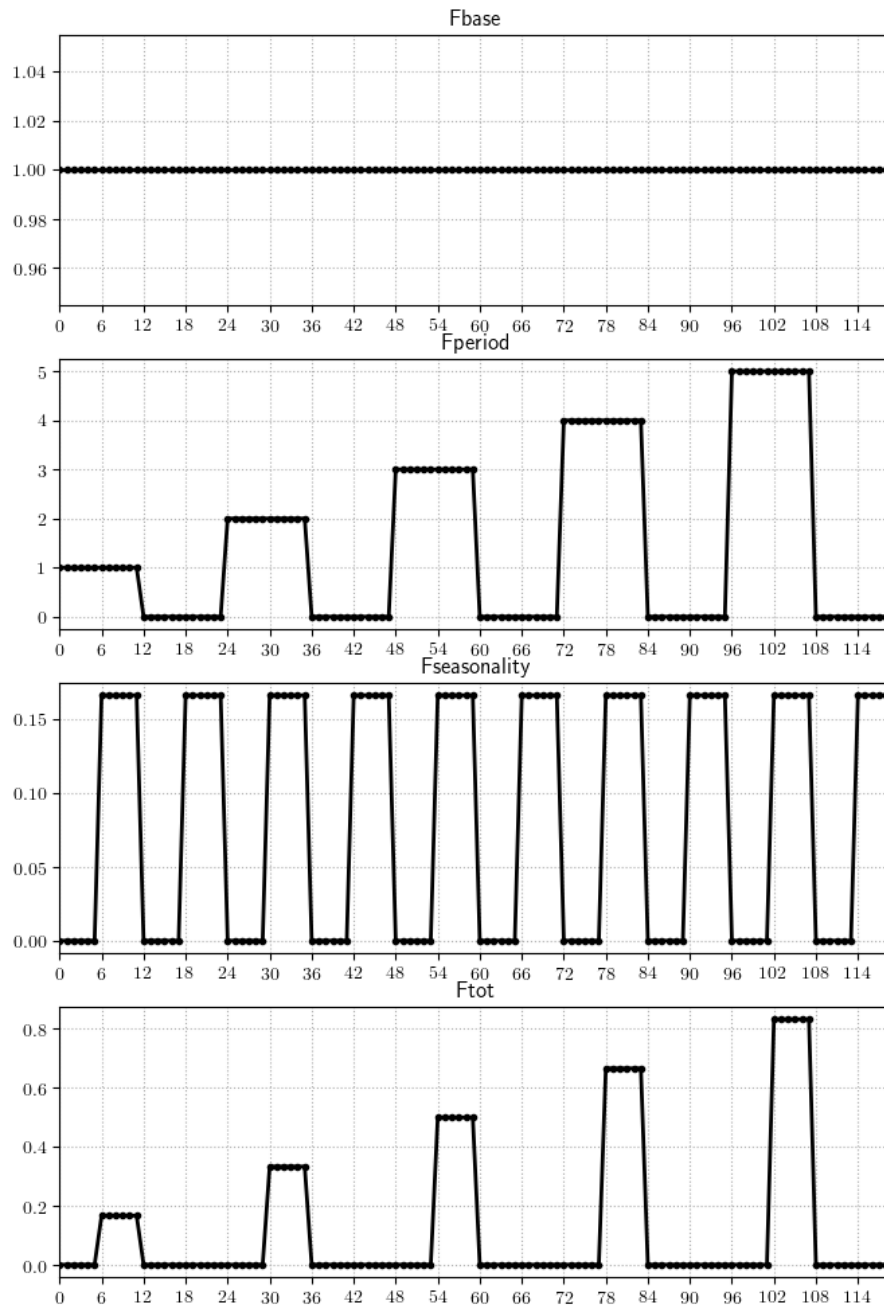
```

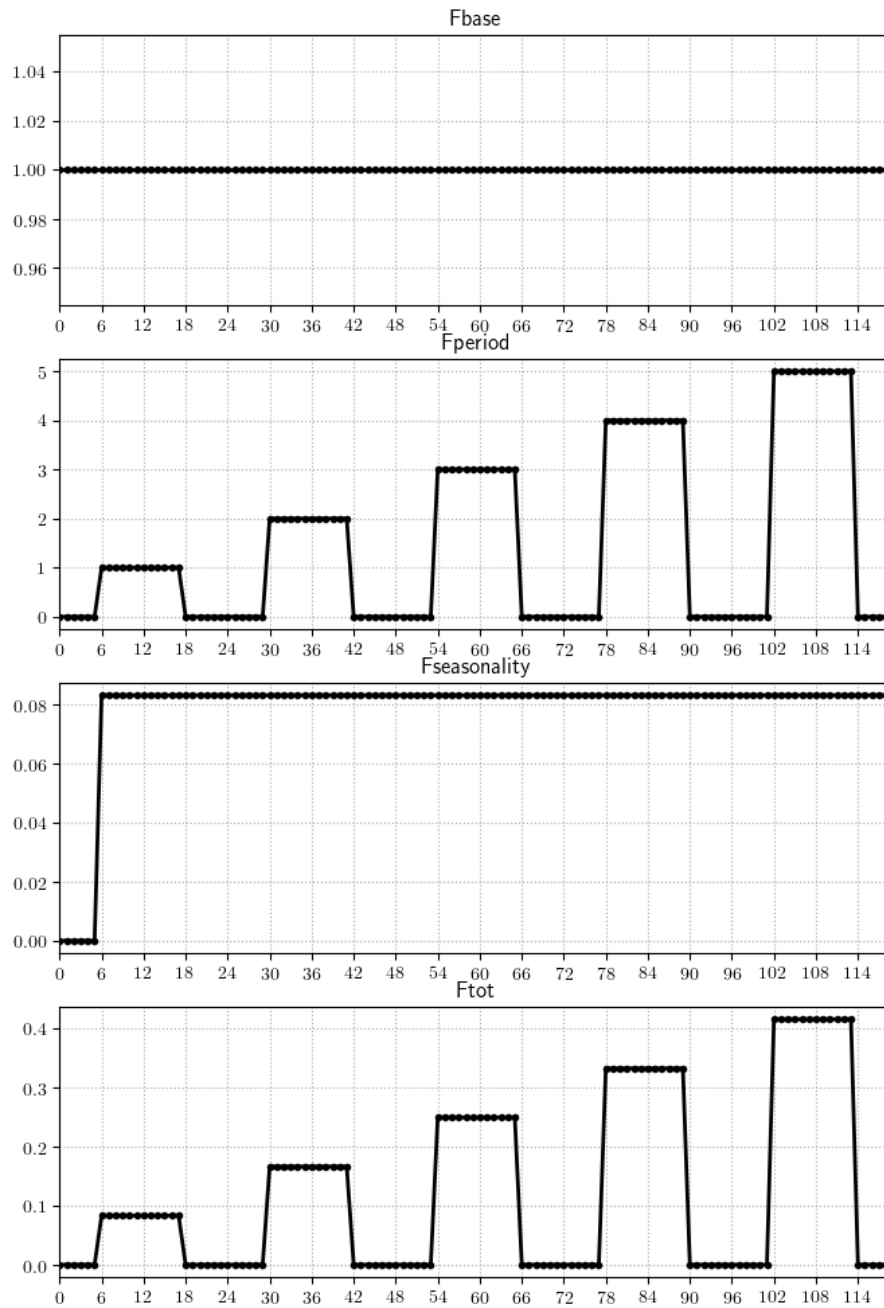
```

fisheries.rate.base.fsh0;1
fisheries.season.number.fsh0;2
fisheries.rate.byperiod.fsh0;0, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0
fisheries.season.start.fsh0;0.25
fisheries.seasonality.fsh0;0.0833;0.0833;0.0833;0.0833;0.0833;0.0833;0.0833;0.
↪0833;0.0833;0.0833;0.0833;

```

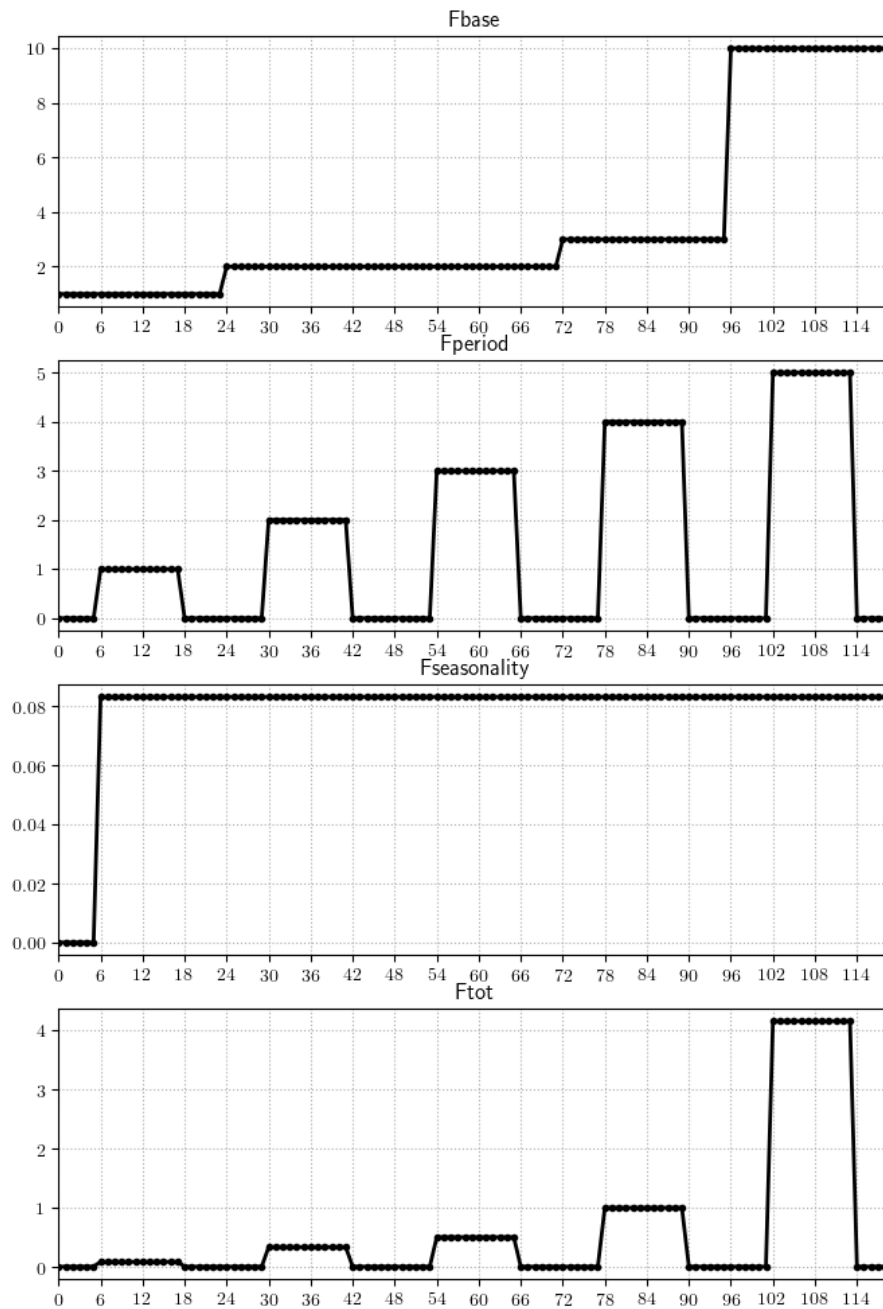




```

fisheries.rate.base.fsh0;1,2,3,10
fisheries.rate.base.shift.fsh0;1, 3, 4
fisheries.season.number.fsh0;2
fisheries.rate.byperiod.fsh0;0, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0
fisheries.season.start.fsh0;0.25
fisheries.seasonality.fsh0;0.0833;0.0833;0.0833;0.0833;0.0833;0.0833;0.0833;0.0833;0.
↪0.0833;0.0833;0.0833;0.0833;

```



```

fisheries.rate.base.fsh0;1,2,3,10

```

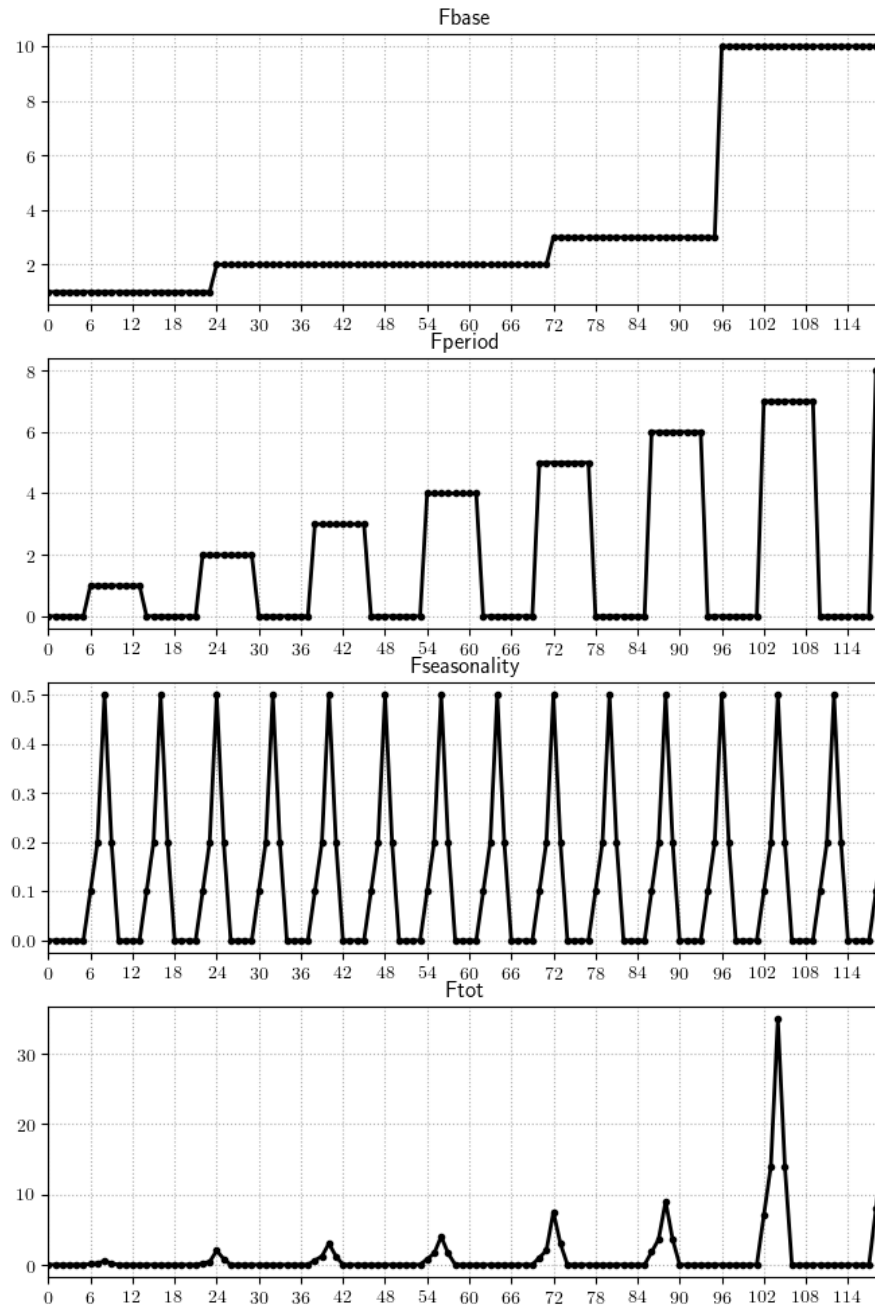
(continues on next page)

(continued from previous page)

```

fisheries.rate.base.shift.fsh0;1, 3, 4
fisheries.season.number.fsh0;3
fisheries.rate.byperiod.fsh0;0, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0, 6, 0, 7, 0, 8
fisheries.season.start.fsh0;0.25
fisheries.seasonality.fsh0;0.1,0.2,0.5,0.2,0,0,0,0

```



Size selectivities

Table 2.40: Fishing size-selectivity parameters

fisheries.selectivity.tiny.fsh#	Selectivities values below which selectivity is forced to 0 (ϵ)
fisheries.selectivity.type.file.fsh#	File containing the selectivity types
fisheries.selectivity.type.shift.fsh#	Array containing the selectivity periods
fisheries.selectivity.type.fsh#	Selectivity types (one value per shift period). Must be 0, 1 or 2
fisheries.selectivity.a50.file.fsh#	File containing the age selectivities.
fisheries.selectivity.a50.shift.fsh#	Array containing the A_{50} shift periods
fisheries.selectivity.a50.fsh#	Age selectivity (one value per shift period). If set, assumes that fishery selectivity is age-based
fisheries.selectivity.l50.file.fsh#	File containing the L_{50} .
fisheries.selectivity.l50.shift.fsh#	Array containing the L_{50} shift periods
fisheries.selectivity.l50.fsh#	L_{50} (one value per shift period).
fisheries.selectivity.l75.file.fsh#	File containing the L_{75}
fisheries.selectivity.l75.shift.fsh#	Array containing the L_{75} shift periods
fisheries.selectivity.l75.fsh#	L_{75} (one value per shift period).

Note that `type`, `a50`, `l50` and `l75` are parameterized in the same way. If the `.file` parameter is defined, then it is used. If it is not set, then values are defined by using the other two parameters. The `shift` array contains thresholds, where the values are to change.

The selectivity type must contain 0 (knife-edge), 1 (sigmoid), 2 (Gaussian) or 3 (log-normal).

If one of the `a50` parameter, it is assumed that age selectivity is used.

Warning: Only knife-edge selectivity can be used with age.

Note: If only knife-edge selectivity is used, then the `l75` parameters are not used.

Knife-edge selectivity

Knife-edge selectivity is computed as follows:

$$S(L) = 1 \text{ if } L \geq L_{50}$$

Sigmoid selectivity

Sigmoid selectivity is computed as follows:

$$S(L) = \frac{1}{1 + \exp^{S_1 - S_2 L}}$$

$$S_1 = \frac{L_{50} \times \ln 3}{L_{75} - L_{50}}$$

$$S_2 = \frac{S_1}{L_{50}}$$

Gaussian selectivity

Gaussian selectivity is computed as follows:

$$S(L) = \frac{F(L)}{F(L_{50})}$$

$$F(L) = \exp\left(-\frac{L - L_{50}}{2\sigma^2}\right)$$

$$\sigma = \frac{L_{75} - L_{50}}{q_{75}}$$

with q_{75} is the inverse cumulative standard normal distribution for the 75th percentile.

Catchability

Fishery catchabilities are parameterized in a similar way as predation accessibility matrix.

Table 2.41: Fisheries catchabilities

fisheries.catchability.file	Name of the catchability file
fisheries.catchability.file.cat#	Name of the catchability file
fisheries.catchability.years.cat#	List of years when the catchability should be used.
fisheries.catchability.initialYear.cat#	First year when the catchability matrix should be used (if year list not provided)
fisheries.catchability.finalYear.cat#	Last year when the catchability matrix should be used (if year list not provided)
fisheries.catchability.steps.cat#	List of steps within a year when the catchability should be used.

Fishery catchabilities should be provided as a CSV file, with fisheries as column (predators) and species (background and focal) as rows (preys). If the first parameter (`fisheries.catchability.file`) is found, then this catchability matrix will be used over the entire simulation.

If this parameter is not found, Osmose will assume that catchability matrixes may vary over time. It will therefore look for all the *fisheries.catchability.file.cat#* parameters. Each catchability matrix should be associated with time-indications, which specifies on which year (interannual variability) and which time-steps (seasonal variability) this catchability matrix should be used.

Warning: Note that the # here is not related to the one of fisheries.

Discards

There is also the possibility to define fisheries discards. It is defined in the same way as catchabilities (cf above for a detailed description of the parameters).

Table 2.42: Fisheries discards

fisheries.discards.file	Name of the catchability file
fisheries.discards.file.dis#	Name of the catchability file
fisheries.discards.initialYear.dis#	First year when the catchability matrix should be used
fisheries.discards.finalYear.dis#	Last year when the catchability matrix should be used
fisheries.discards.years.dis#	List of years when the catchability should be used.
fisheries.discards.steps.dis#	List of steps within a year when the catchability should be used.

Fishery discards should be provided as a CSV file, with fisheries as column (predators) and species (background and focal) as rows (preys).

PRE/POST PROCESSING OF OSMOSE MODEL

In this section, the pre/post processing tools of the `osmose` package are presented

3.1 Library loading

```
require("osmose")  
  
ls("package:osmose")
```

```
Loading required package: osmose  
[1] "cacheManager"      "get_var"           "getVar"  
[4] "osmose_calib_demo" "osmose_demo"       "osmose2R"  
[7] "read_osmose"        "report"            "run_osmose"  
[10] "runOsmose"          "update_osmose"     "write_osmose"  
[13] "write.osmose"
```

3.2 Reading Osmose parameters

The loading of Osmose parameters is achieved by using the `read_osmose` function with an `input` argument providing the path of the main configuration file.

```
suppressMessages(require("osmose"))  
  
output.dir = file.path("rosmose", "_static")  
  
exampleFolder <- tempdir()  
demoPaths <- osmose_demo(path = exampleFolder, config = "eec_4.3.0")  
  
# load the Osmose configuration file  
osmConf = read_osmose(input=demoPaths$config_file)  
  
# Summarize the configuration  
summary(osmConf)  
  
# extracting a particular configuration object  
sim = get_var(osmConf, what="simulation")
```

(continues on next page)

(continued from previous page)

```
# proper way to recover the season parameter (returns a vector of numeric)
movements = get_var(osmConf, what='movement')

# Plotting reproduction growth
png(filename = file.path(output.dir, 'species.png'))
plot(osmConf, what="species", species=0)

# Plotting reproduction seasonality
png(filename = file.path(output.dir, 'reproduction.png'))
plot(osmConf, what="reproduction", species=1)

# Plotting size range for predation
```

	Length	Class	Mode
osmose	2	-none-	list
mortality	4	-none-	list
simulation	6	-none-	list
movement	7	-none-	list
fisheries	9	-none-	list
predation	4	-none-	list
reproduction	1	-none-	list
species	16	-none-	list
grid	3	-none-	list
population	1	-none-	list
output	18	-none-	list

Furthermore, some parameters can be plotted for a given species. This is done as follows:

```
png(filename = file.path(output.dir, 'predation.png'))
plot(osmConf, what="predation", species=2)
```

3.3 Running the model

OSMOSE Java can be run from R by using the `run_osmose` function.

```
library("osmose")

# recover the reference configuration file
filename = system.file(package="osmose", "extdata", "master", "osm_all-parameters.csv")

# setting output directory
outdir = 'output'

# default run mode (java file in inst/java/)
run_osmose(input=fileName, output=outdir)
```

By default, the OSMOSE Java included in the package is used. However, the user is free to use another build of the OSMOSE Java program as follows:

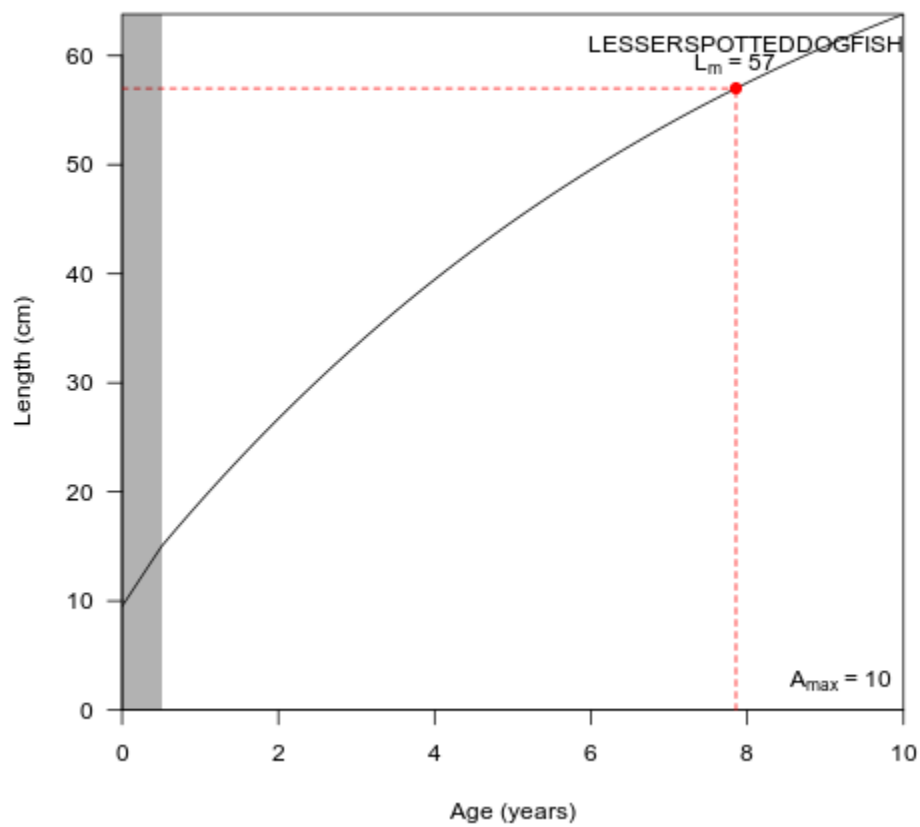


Fig. 3.1: Growth parameters for species 0

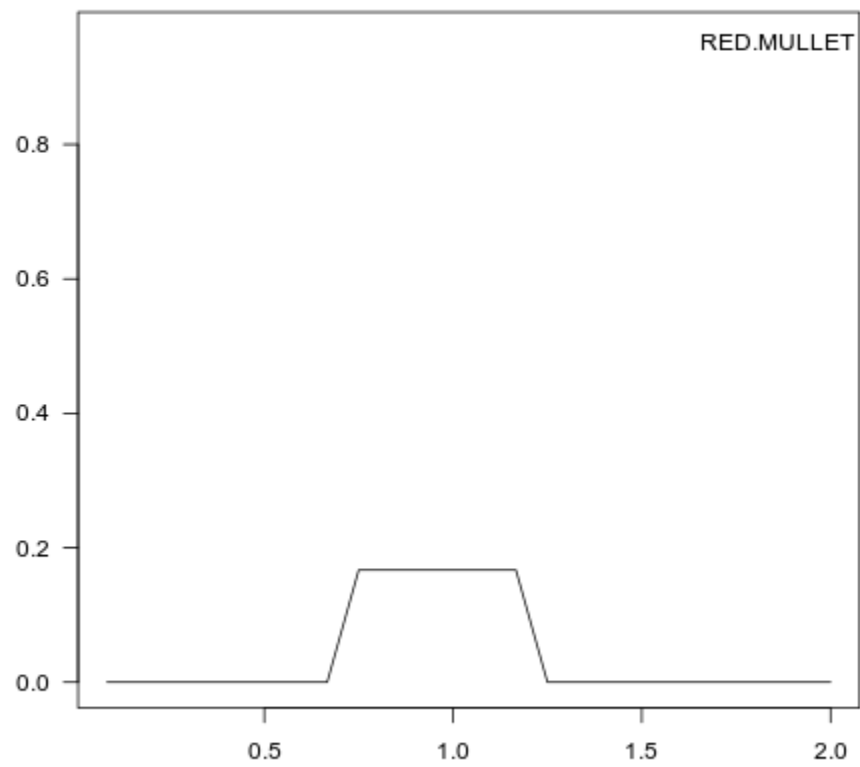


Fig. 3.2: Reproduction seasonality for species 1

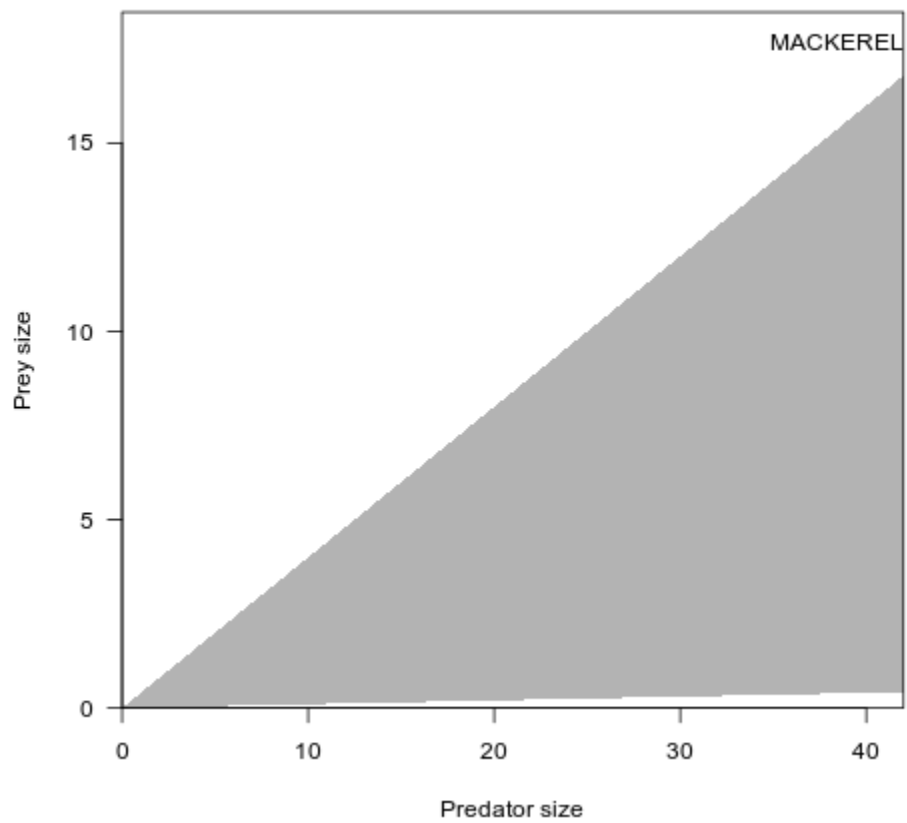


Fig. 3.3: Predation size range for species 2

```
# Running Osmose by using another version of the .jar file.
jarfile = "/home/nbarrier/Modeles/osmose/svn-osmose/trunk/dist/osmose-trunk.jar"
run_osmose(osmose=jarfile, input=fileName,
           output=outdir, version="4")
```

Warning: The way the OSMOSE Java core is called has changed between versions 3 and 4. As a consequence, the user must be sure to properly set the version argument.

3.4 Reading outputs

The reading of outputs is achieved by using the `read_osmose` function:

```
library("osmose")

data = read_osmose(path='output')
```

The content of the data object can be obtained as follows:

```
names(data)
```

[1]	"model"	"species"	"biomass"
[4]	"abundance"	"mortality"	"meanTL"
[7]	"meanTLCatch"	"biomassByTL"	"predatorPressure"
[10]	"predPreyIni"	"dietMatrix"	"meanSize"
[13]	"meanSizeCatch"	"SizeSpectrum"	"abundanceBySize"
[16]	"biomassBySize"	"meanTLBySize"	"mortalityBySize"
[19]	"dietMatrixBySize"	"predatorPressureBySize"	"abundanceByAge"
[22]	"biomassByAge"	"meanSizeByAge"	"meanTLByAge"
[25]	"mortalityByAge"	"dietMatrixByAge"	"predatorPressureByAge"
[28]	"abundanceByTL"	"yieldByFishery"	"yield"
[31]	"yieldN"	"yieldBySize"	"yieldNBySize"
[34]	"yieldByAge"	"yieldNByAge"	"sizeMature"
[37]	"ageMature"	"ingestion"	"ingestionTot"
[40]	"maintenance"	"meanEnet"	"sizeInf"
[43]	"kappa"	"abundAge1"	"ingestByAge"
[46]	"ingestBySize"	"kappaByAge"	"kappaBySize"
[49]	"enetByAge"	"enetBySize"	"maintenanceByAge"
[52]	"maintenanceBySize"	"config"	

Variables are accessed by using the `getVar` function, which allows to apply some operations prior to extraction (average over the replicates, conversion into a list or a matrix):

```
library("osmose")

getwd()
input.dir = file.path("eec_4.3.0", "output-PAPIER-trophic")
data = read_osmose(input.dir)

biom = get_var(data, "biomass")
```

(continues on next page)

(continued from previous page)

```

class(biom)
dim(biom)
cat("\n")

# expected = TRUE if mean over replicate should be returned
# only works for data that inherits from array
biom = get_var(data, "biomass", expected=TRUE)
class(biom)
dim(biom)
cat("\n")

biom = get_var(data, "biomass", how="list")
class(biom)
names(biom)
cat(biom$OctopusVulgaris)
cat("\n")

biom = get_var(data, "dietMatrixByAge")
class(biom)
cat("\n")

biom = get_var(data, "dietMatrixByAge", how="list")
class(biom)

```

```

[1] "/home/barrier/Work/codes/osmose/osmose-doc"
[1] "osmose.biomass" "array"
[1] 8 14 3

[1] "matrix" "array"
[1] 8 14

[1] "list"
[1] "lesserSpottedDogfish" "redMullet" "pouting"
[4] "whiting" "poorCod" "cod"
[7] "dragonet" "sole" "plaice"
[10] "horseMackerel" "mackerel" "herring"
[13] "sardine" "squids"

[1] "osmose.dietMatrixByAge" "list"

[1] "list"

```

The first argument is the data object obtained by using the `read_osmose` function, while the second argument is the name of the variable to extract.

3.4.1 Reading NetCDF outputs

In the new Osmose version, the user has the possibility to save outputs as NetCDF (.nc) instead of CSV (.csv) files. **However, no function has been provided to automatically read all the Osmose NetCDF files.** Therefore, scripts must be written by the user, following the example shown below:

```
rm(list=ls())

library("ncdf4")

filename = file.path("rosmose", "_static",
                     "gogosm_mortalityRateDistribByAge-OctopusVulgaris_Simu0.nc")

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ fid")
fid = nc_open(filename)
print(fid)

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ vars")
var = fid$var
print(names(var))
mort = ncvar_get(fid, "mortality")

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ dims")
dims = fid$dim
print(names(dims))

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mortality")
mcause = ncvar_get(fid, "mortality_cause")
attrs = ncatt_get(fid, "mortality_cause")
print(attrs)

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ age")
age = ncvar_get(fid, "Age")
print(age)
```

```
[1] "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ fid"
File rosmose/_static/gogosm_mortalityRateDistribByAge-OctopusVulgaris_Simu0.nc (NC_
↳FORMAT_CLASSIC):

    1 variables (excluding dimension variables):
        float mortality[mortality_cause,Age,time]
            units:
                description: Predation (Mpred), Starvation (Mstarv), Other Natural mortality_
↳(Mnat), Fishing (F) & Out-of-domain (Z) mortality rates per time step of saving and_
↳per size class. Z is the total mortality for migratory fish outside the simulation_
↳grid. To get annual mortality rates, sum the mortality rates within one year.
                _FillValue: -99

    3 dimensions:
        time Size:12 *** is unlimited ***
            units: days since 0-1-1 0:0:0
            calendar: 360_day
            description: time elapsed, in days, since the beginning of the simulation
```

(continues on next page)

(continued from previous page)

```

Age Size:25
mortality_cause Size:6
  mortality_cause0: PREDATION
  mortality_cause1: STARVATION
  mortality_cause2: ADDITIONAL
  mortality_cause3: FISHING
  mortality_cause4: OUT
  mortality_cause5: OXY
[1] "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa vars"
[1] "mortality"
[1] "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa dims"
[1] "time"           "Age"           "mortality_cause"
[1] "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa mortality"
$mortality_cause0
[1] "PREDATION"

$mortality_cause1
[1] "STARVATION"

$mortality_cause2
[1] "ADDITIONAL"

$mortality_cause3
[1] "FISHING"

$mortality_cause4
[1] "OUT"

$mortality_cause5
[1] "OXY"

[1] "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa age"
[1] 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

```

3.5 Plotting outputs

Plotting Osmose outputs is achieved by calling the generic plot function on the `osmose` object, output by the `read_osmose` function. The function takes as argument the variable to be displayed (`what` argument). The `species` argument allows to specify the indexes of the species to display (cf. the `X` in the `species.name.spX` parameters).

```

library("osmose")

input.dir = file.path("eec_4.3.0", "output-PAPIER-trophic")
data = read_osmose(input.dir)

output.dir = file.path("rosmose", "_static")

test = get_var(data, "species")

png(file=file.path(output.dir, "biomass.png"))

```

(continues on next page)

(continued from previous page)

```

plot(data, what="biomass")

png(file=file.path(output.dir, "abundance.png"))
plot(data, what="abundance", species=c(0, 1, 2))

png(file=file.path(output.dir, "yield.png"))
plot(data, what="yield")

png(file=file.path(output.dir, "yieldN.png"))
plot(data, what="yieldN")

```

Hint: When several replicates are run, the uncertainty due to the stochastic mortality is displayed as a grey shading.

Warning: At this time, only these four variables can be plotted. However, more plot functions (diet matrix, mortality, etc.) will be released soon.

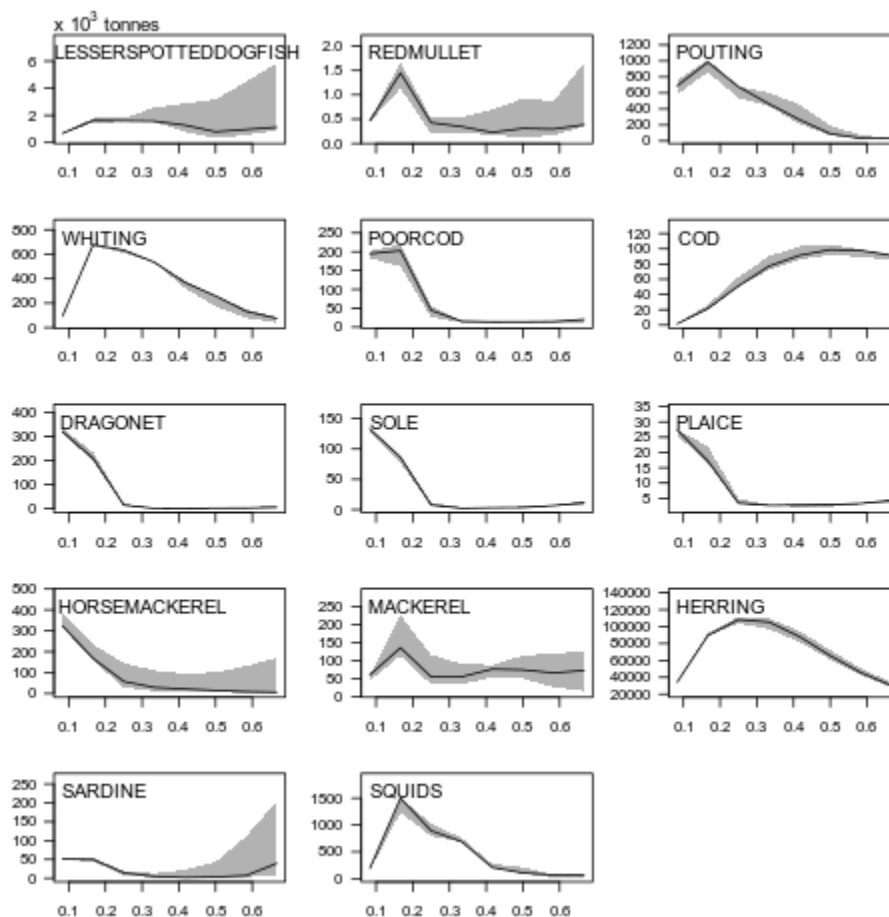


Fig. 3.4: Biomass plot

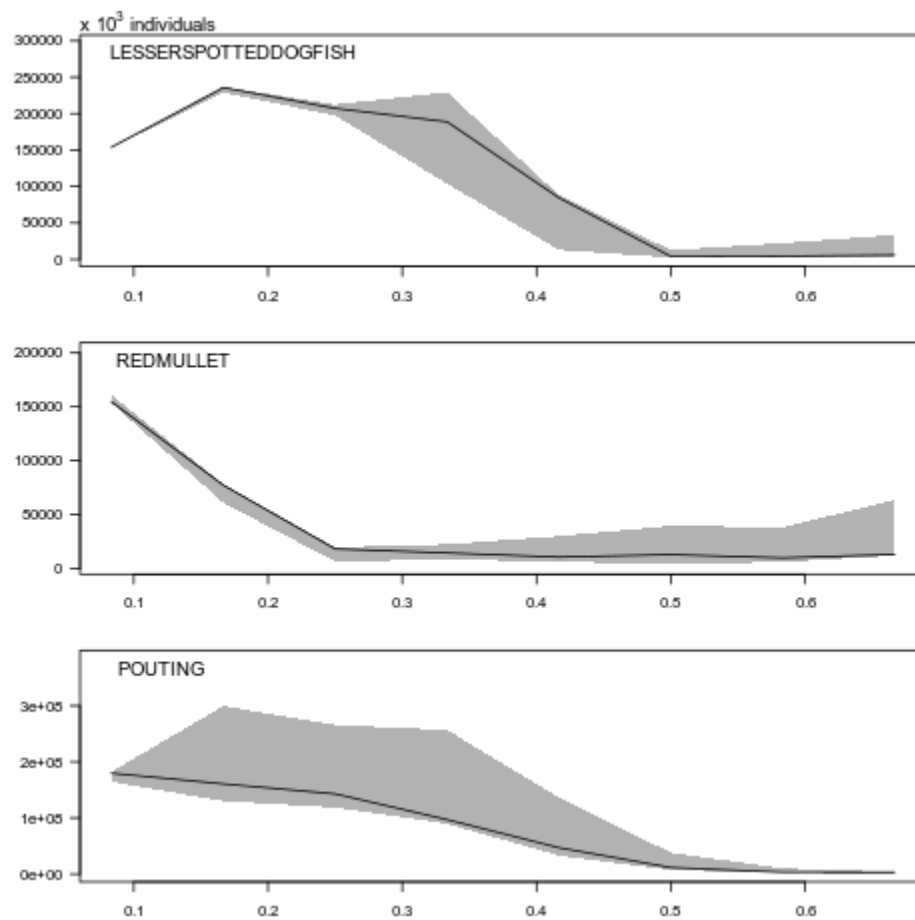


Fig. 3.5: Abundance plot

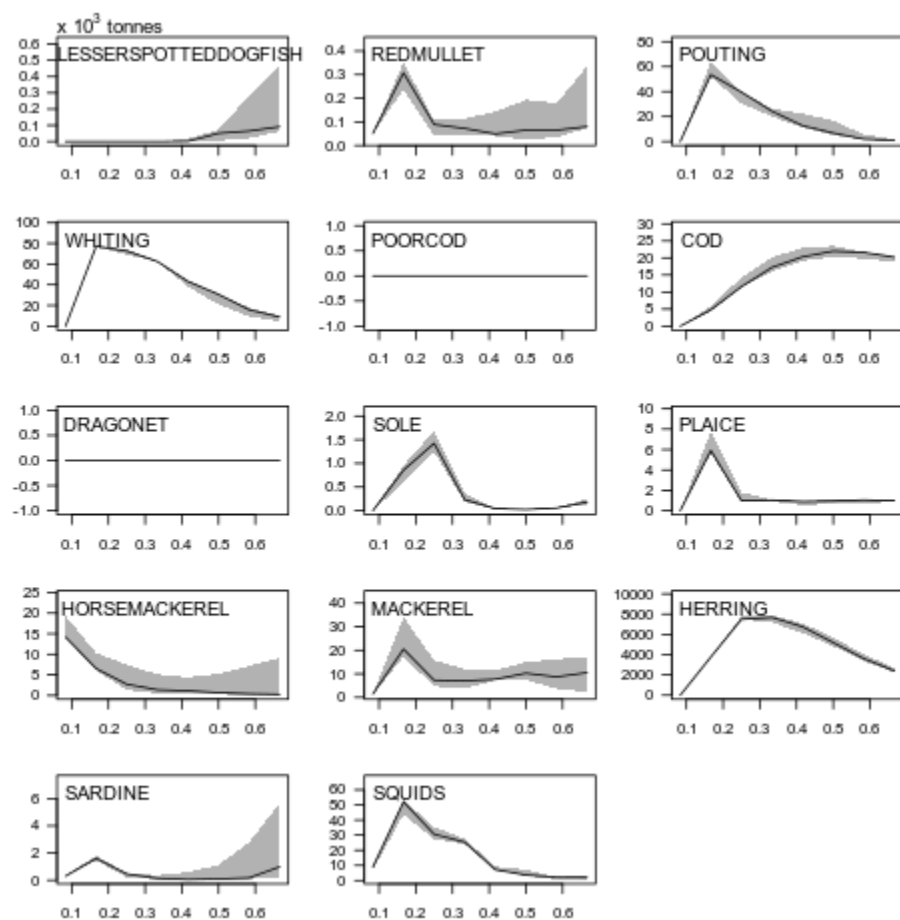


Fig. 3.6: Yield biomass plot

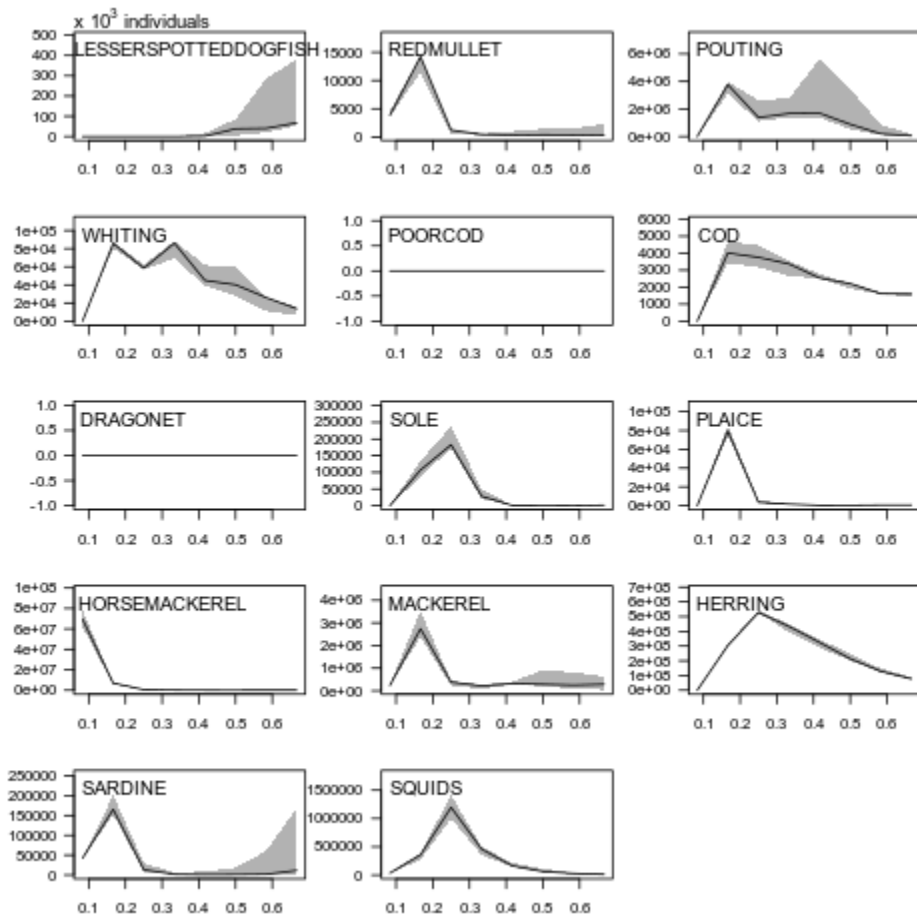


Fig. 3.7: Yield abundance plot

RUNNING A CALIBRATION

In this section, the methodology of OSMOSE calibration using the sequential approach described in [ORVES17] is provided. This calibration uses the `calibrar` and `osmose` R libraries (see [Section 1](#) for installation instructions). For a detailed description of the `calibrar` package, the user is referred to [ORS17].

The philosophy of the method is to assess unknown or poorly described parameters (for instance larval mortality or plankton accessibility) by fitting model outputs to observed datasets (biomass, etc.).

The calibration of the reference calibration, which is explained in details in the following, can be tested by typing in R:

```
rm(list=ls())

library("osmose")

# copy all that is needed for the calibration
demo = osmose_calib_demo(path = "calib/_static/")

# move to the calibration folder
setwd(demo$path)

# run the calibration script
#source(demo$file)
```

```
In [1]: import os

In [2]: import subprocess

In [3]: cwd = os.getcwd()

In [4]: fpath = "calib/_static/democalib.R"

In [5]: with open(fpath) as f:
...:     with open(os.devnull, "w") as DEVNULL:
...:         subprocess.call(["Rscript", fpath])
...:
```

4.1 Calibration directory architecture

To run the OSMOSE calibration, it is assumed that the following directories and files exist:

- a directory containing observation data used in the calibration. In this example: `DATA` (see [Section 4.2](#))
- a directory containing the configuration and forcing files that will be copied in the individual run directories of each population (in this example, `master`).

In this master directory, the configuration file that will be used in the calibration process must contain the following two lines:

Table 4.1: Osmose main configuration file

<code>osmose.configuration.calibration</code>	<code>calibration-parameters.csv</code>
<code>osmose.configuration.main</code>	<code>osm_all-parameters.csv</code>

The first entry is the name of the CSV file that will contain the calibrated parameters. This file must be generated in the calibration process (see [Section 4.6](#)).

The second entry is the path of the main configuration file, which contains all the uncalibrated parameters needed by OSMOSE Java.

Danger: The main configuration file (here, `osm_all-parameters.csv`) may also contain the calibrated parameters (defined in `calibration-parameters.csv`). **In this case, the `calibration-parameters.csv` file must be included first, in order to insure that it will have precedence over the `osm_all-parameters.csv` file.**

4.2 Data preparation

Data used to calibrate the OSMOSE model must be on CSV format. Data may contain one or more columns.

Two examples are provided below.

Table 4.2: Annual data biomass: `EngraulisEncrasicolus.biomass.csv`

	<code>EngraulisEncrasicolus.biomass</code>
1	13000
2	13000
3	13000
4	13000
5	13000
6	13000
7	13000
8	13000
9	13000
10	13000
11	13000
12	13000

Table 4.3: Monthly data thresholds: MustelusMustelus.thr.csv

	min	max
1	804	3216
2	804	3216
3	804	3216
4	804	3216
5	804	3216
6	804	3216
7	804	3216
8	804	3216
9	804	3216
10	804	3216
11	804	3216
12	804	3216
13	804	3216
14	804	3216
15	804	3216
16	804	3216
17	804	3216
18	804	3216
19	804	3216
20	804	3216
21	804	3216
22	804	3216
23	804	3216
24	804	3216
25	804	3216
26	804	3216
27	804	3216
28	804	3216
29	804	3216
30	804	3216
31	804	3216
32	804	3216
33	804	3216
34	804	3216
35	804	3216
36	804	3216
37	804	3216
38	804	3216
39	804	3216
40	804	3216
41	804	3216
42	804	3216
43	804	3216
44	804	3216
45	804	3216
46	804	3216
47	804	3216
48	804	3216
49	804	3216

continues on next page

Table 4.3 – continued from previous page

50	804	3216
51	804	3216
52	804	3216
53	804	3216
54	804	3216
55	804	3216
56	804	3216
57	804	3216
58	804	3216
59	804	3216
60	804	3216
61	804	3216
62	804	3216
63	804	3216
64	804	3216
65	804	3216
66	804	3216
67	804	3216
68	804	3216
69	804	3216
70	804	3216
71	804	3216
72	804	3216
73	804	3216
74	804	3216
75	804	3216
76	804	3216
77	804	3216
78	804	3216
79	804	3216
80	804	3216
81	804	3216
82	804	3216
83	804	3216
84	804	3216
85	804	3216
86	804	3216
87	804	3216
88	804	3216
89	804	3216
90	804	3216
91	804	3216
92	804	3216
93	804	3216
94	804	3216
95	804	3216
96	804	3216
97	804	3216
98	804	3216
99	804	3216

continues on next page

Table 4.3 – continued from previous page

100	804	3216
101	804	3216
102	804	3216
103	804	3216
104	804	3216
105	804	3216
106	804	3216
107	804	3216
108	804	3216
109	804	3216
110	804	3216
111	804	3216
112	804	3216
113	804	3216
114	804	3216
115	804	3216
116	804	3216
117	804	3216
118	804	3216
119	804	3216
120	804	3216

4.3 Calibration settings

The user must create a CSV file that contains informations about the data that will be used in the calibration process (see [Table 4.4](#)). This file will be read by the `calibrar` package.

Table 4.4: Calibration settings

variable	type	calibrate	weights	useData
TrachurusTrachurus.biomass	lnorm2	TRUE	12.5	TRUE
SardinaPilchardus.biomass	lnorm2	TRUE	12.5	TRUE
SardinellaAurita.biomass	lnorm2	TRUE	12.5	TRUE
EngraulisEncrasicolus.biomass	lnorm2	TRUE	12.5	TRUE
OctopusVulgaris.landings	lnorm2	TRUE	22.22	TRUE
MelicertusKerathurus.landings	lnorm2	TRUE	22.22	TRUE
MetapenaeusMonoceros.landings	lnorm2	TRUE	22.22	TRUE
TrachurusTrachurus.landings	lnorm2	TRUE	22.22	TRUE
SardinaPilchardus.landings	lnorm2	TRUE	22.22	TRUE
SardinellaAurita.landings	lnorm2	TRUE	22.22	TRUE
EngraulisEncrasicolus.landings	lnorm2	TRUE	22.22	TRUE
DiplodusAnnularis.landings	lnorm2	TRUE	22.22	TRUE
MustelusMustelus.landings	lnorm2	TRUE	22.22	TRUE
MerlucciusMerluccius.landings	lnorm2	TRUE	22.22	TRUE
PagellusErythrinus.landings	lnorm2	TRUE	22.22	TRUE
OctopusVulgaris.thr	minmaxt	TRUE	1	TRUE
MelicertusKerathurus.thr	minmaxt	TRUE	1	TRUE
MetapenaeusMonoceros.thr	minmaxt	TRUE	1	TRUE
TrachurusTrachurus.thr	minmaxt	TRUE	1	TRUE
SardinaPilchardus.thr	minmaxt	TRUE	1	TRUE
SardinellaAurita.thr	minmaxt	TRUE	1	TRUE
EngraulisEncrasicolus.thr	minmaxt	TRUE	1	TRUE
DiplodusAnnularis.thr	minmaxt	TRUE	1	TRUE
MustelusMustelus.thr	minmaxt	TRUE	1	TRUE
MerlucciusMerluccius.thr	minmaxt	TRUE	1	TRUE
PagellusErythrinus.thr	minmaxt	TRUE	1	TRUE

The table columns are:

- **variable** is the list of the variables that will be fit the model with observations.
- **type** is the likelihood function that will be used in the fitting. It can either be a native function of the `calibrar` package or a user defined function.
- **calibrate** indicates whether the data is used in the calibration.
- **weights** provides the relative weights used to combine the partial objective values obtained for each variable.
- **useData** indicates whether data are read from the disk. If `useData=FALSE`, the observed value is set to `NULL` and the likelihood function is expected to use simulated data only. The latter option can be particularly useful to set penalties in the model outputs or parameters, where no observed data are needed.

Warning: If `useData = TRUE`, the name of the CSV file must be consistent with the name of the variable. For instance, if the variable is `PagellusErythrinus.thr`, the data file must be `PagellusErythrinus.thr.csv`

Danger: The column names (i.e. the header) are important and must be as shown in [Table 4.4](#).

4.4 List of variables to calibrate

The next step is to create a table containing the list of the Osmose parameters to calibrate (see [Table 4.5](#)).

Table 4.5: Parameters to calibrate

	paropt	parmin	parmax	parphase
mortality.natural.larva.rate.sp0	0.476013813	0.2	1.6	2
mortality.natural.larva.rate.sp1	1.578997873	0.6	2.0	2
mortality.natural.larva.rate.sp2	1.19819857	0.4	1.6	2
mortality.natural.larva.rate.sp3	0.586048506	0.2	1.2	2
mortality.natural.larva.rate.sp4	0.2	0.2	1.2	2
mortality.natural.larva.rate.sp5	0.904776806	0.4	1.6	2
mortality.natural.larva.rate.sp6	0.859194349	0.4	1.6	2
mortality.natural.larva.rate.sp7	1.281004774	0.4	1.6	2
mortality.natural.larva.rate.sp8	0.010436663	0.005	0.02	2
mortality.natural.larva.rate.sp9	0.416166226	0.2	1.2	2
mortality.natural.larva.rate.sp10	1.530131413	0.6	1.8	2
plankton.accessibility2fish.plk0	0.1	0.05	1.9	1
plankton.accessibility2fish.plk1	0.1	0.05	1.9	1
plankton.accessibility2fish.plk2	0.05	0.05	1.9	1
plankton.accessibility2fish.plk3	0.1	0.05	1.9	1
plankton.accessibility2fish.plk4	0.1	0.05	1.9	1
plankton.accessibility2fish.plk5	1.45	0.05	1.9	1
mortality.fishing.rate.sp0	0.130877451	0.1	0.5	3
mortality.fishing.rate.sp1	0.382709506	0.175	0.7	3
mortality.fishing.rate.sp2	0.499612516	0.227	0.908	3
mortality.fishing.rate.sp3	0.2	0.125	0.5	3
mortality.fishing.rate.sp4	0.6	0.175	0.7	3
mortality.fishing.rate.sp5	0.5	0.125	0.6	3
mortality.fishing.rate.sp6	0.006662997	0.003	0.012	3
mortality.fishing.rate.sp7	0.6	0.225	0.9	3
mortality.fishing.rate.sp8	0.15	0.037723225	0.2	3
mortality.fishing.rate.sp9	0.09638775	0.05	0.2	3
mortality.fishing.rate.sp10	0.5	0.2	0.694	3
q.sp0	0.959522483	0.5	1.3	4
q.sp1	1.007415552	0.5	1.3	4
q.sp2	0.998406705	0.5	1.3	4
q.sp3	0.897712618	0.5	1.3	4
q.sp4	0.936632054	0.5	1.3	4
q.sp5	0.929764322	0.5	1.3	4
q.sp6	0.838034913	0.5	1.3	4
q.sp7	0.848811769	0.5	1.3	4
q.sp8	0.956199768	0.5	1.3	4
q.sp9	0.948716213	0.5	1.3	4
q.sp10	0.829510129	0.5	1.3	4

This table is as follows:

- the first column contains the parameters to calibrate.
- paropt is the parameter initial guess. If NA, the values are initialized as the mean of parmin and parmax (if both are defined). Else, initialized as 0.

- `parmin` is the minimum possible value of the parameter
- `parmax` is the maximum possible value of the parameter
- `parphase` is the calibration phase in which the parameter will be calibrated. If `NA`, the parameter is not calibrated.

Hint: Here, the table format do not matter since this table will be read by the user prior to run the calibration. **However, it is recommended to keep this format.**

In this example, all the parameters except the `q.spX` ones, are OSMOSE Java parameters. The `q.spX` parameters can be viewed as “capturability indexes”. Since observation datasets may not be on the same spatial scale as model outputs (survey zone, etc.), this index is used to convert Osmose outputs into observational dataset.

Danger: The names of the OSMOSE Java calibrated parameters must be properly set, in order to be insure that they will be used by the model.

4.5 Create the calibration script

The next step is to write the calibration script, which is the main script to execute to run the calibration.

The sample script used to calibrate the [HLS+16] configuration, provide in the `osmose` package. is detailed below.

Danger: All the files and script provided in the package are built for the calibration of the reference configuration. They should never be used to calibrate another configuration.

4.5.1 Library and function loading

Load the libraries and source the `runModel.R` file (which must be written by the user, see [Section 4.6](#)).

```
rm(list=ls())
require("osmose")
require("calibrar")
require("doParallel")

# Recovering of the runModel function associated with the reference_
↳ configuration
dirin = getwd()
filename = file.path(dirin, "runModel.R")
source(filename)
```

4.5.2 Define your likelyhood functions

The user has the possibility to define its own likelyhood function. For instance:

```
# creates a user defined likelyhood function
minmaxt = function(obs, sim) {
  output = -1e+6*sum(log(pmin((sim+1)/(obs[, 1]+1), 1)), na.rm=TRUE) +
           1e+6*sum(log(pmax((sim+1)/(obs[, 2]+1), 1)), na.rm=TRUE)
  return(output)
}
```

Warning: The function name must match the name of the `type` column in the calibration settings (see [Table 4.4](#))

4.5.3 Read calibration information

The next step is to read the calibration informations (see [Table 4.4](#)). This is done by using the `calibrar::getCalibrationInfo` function.

```
# reads calibration informations
calib_path = dirin
calib_file = "calibration_settings.csv"
calInfo = getCalibrationInfo(path=calib_path, file=calib_file)
```

4.5.4 Read observation datasets

The next step is to load the observation datasets by using the `calibrar::getObservedData` function.

```
# loads the observed data (path is the data directory path,
# data.folder is the data directory name)
observed = getObservedData(calInfo, path=calib_path, data.folder="DATA")
```

4.5.5 Load the calibrated parameters

Load the calibrated parameters (see [Table 4.5](#)).

In this example, the reading of the parameters is done as follows:

```
# load calibration parameters
param_to_calib = file.path(dirin, "parameters_to_calib.csv")
calibData = read.csv(file=param_to_calib,
                    header=TRUE,
                    sep=",",
                    row.names=1)
```

Warning: Since the format of this file is free, the user must insure that it is properly read.

4.5.6 Create objective function

The next step is to create the objective function, by using the `calibrar::createObjectiveFunction` function. It's arguments are:

- `runModel`: the function that is used to run the model (which must be created by the user, see [Section 4.6](#))
- `info`: the calibration information (output of the `calibrar::getCalibrationInfo` function)
- `observed`: the observed datasets (output of the `calibrar::getObservedData` function)
- `aggregate`: if `TRUE`, a scalar value is returned by using the `aggFn`.
- `aggFn`: the aggregation function. Default is a weighted sum (the weights being provided in [Table 4.4](#)).
- the last arguments are the additional arguments of the `runModel` function (in this case, the `names` arguments)

```
# create an objective function
# additional arguments to the runModel function
# are provided here.
objfn = createObjectiveFunction(runModel=runModel,
                               info=calInfo,
                               observed=observed,
                               aggFn=calibrar:::.weighted.sum,
                               aggregate=FALSE,
                               names=row.names(calibData))
```

4.5.7 Run the calibration

Finally, the calibration is run by using the `calibrate` function.

```
control = list()
# control$maxgen = c(2, 2, 2, 2) # maximum number of generations (former gen.max_
# parameter)
control$maxgen = 2 # maximum number of generations (former gen.max parameter)
control$master = "gog" # directory that will be copied
control$run = "RUN" # run directory
control$restart.file = "calib_restart" # name of the restart file
control$REPORT = 1 # number of generations to run before saving a restart
control$parallel = TRUE
control$nCores = 2
control$popsize = 15 # population size (former seed parameter)

cl = makeCluster(control$nCores)
registerDoParallel(cl)

# send the variables and loaded libraries defined in the above to the nodes
clusterExport(cl, c("objfn", "calibData", "calInfo", "observed", "minmaxt"))
clusterEvalQ(cl, library("osmose"))
clusterEvalQ(cl, library("calibrar"))

cal1 = calibrate(calibData['paropt'], fn=objfn, method='default',
                 lower=calibData['parmin'], upper=calibData['parmax'],
                 phases=calibData['parphase'], replicates=1, control=control)

stopCluster(cl)
```


The arguments are:

- the parameter initial guess
- `fn`: the objective function
- `method`: the optimization method. The default method is the *Adaptive Hierarchical Recombination Evolutionary Strategy (AHR-ES)*
- `lower`: the lower parameter bounds
- `upper`: the upper parameter bounds
- `phases`: the calibration phase at which the parameters are estimated.
- `replicates`: the number of replicates (i.e the number of times the `runModel` function will be called).
- `control`: additional arguments stored in a list (see [Table 4.6](#)).

Table 4.6: Elements that can be put in the `control` argument list

Option	Description
<code>maxit</code>	Maximum number of executions of the objective function.
<code>maxgen</code>	Maximum number of generations
<code>popsize</code>	Population size
<code>parallel</code>	Boolean, TRUE in order to activate the parallel execution of the optimization.
<code>ncores</code>	The number of cores available in the parallel cluster for the active session. If <code>parallel=TRUE</code> , the default is to get the number of cores of the system.
<code>run</code>	An optional folder path to create all the temporary folders needed to run the simulations for each parameter combination tested by the optimization algorithm. The folders are recycled every generation
<code>master</code>	An optional folder path. All the contents of the designated folder will be copied to each temporary folder.
<code>REPORT</code>	Number of iterations after saving a new restart object, which contains all the information necessary to restart the calibration at that point. The default is NULL, and no restart files are created.
<code>restartfile</code>	Filename for the restart file to be created.

In the above example, calibration for each population will be performed in a `RUN` directory. For each population, a `RUN/iX` will be created, which will be the working directory in which the `runModel` function will be called.

The `master` directory (containing the `calibration-parameters.csv` and `config.csv` configuration files) will be copied in all the `RUN/iX` directories.

Hint: For OSMOSE calibration, you can control the number of replicates either through the `replicates` argument, or by setting the OSMOSE Java parameter `simulation.nsimulation`.

Note: Either `maxit` or `maxgen` must be provided. If `maxit` is provided, `maxgen` is computed as

$$\frac{\text{maxit}}{\text{replicates} \times \text{popsize}}$$

Warning: Since the OSMOSE model is stochastic, gradient base methods are inappropriate. The only method compatible with OSMOSE are AHR-ES.

4.5.8 Running calibration in parallel

Because the calibration process is time and resource consuming, it is highly advised to run the calibration in parallel. This is achieved by setting the `control$parallel = TRUE` and by initializing the cluster prior the call to the `calibrate` function.

Furthermore, the content that is used in the calibration (calibration data, likelihood functions, packages) must be exported to the cluster. This is done by using the `clusterExport` and `clusterEvalQ` functions. The `calibrate.R` file must be edited as follows:

```
control$parallel = TRUE
control$nCores = 2 # default value is the available number of cores

cl = makeCluster(control$nCores)
registerDoParallel(cl)

# send the variables and loaded libraries defined in the above to the nodes
clusterExport(cl, c("objfn", "calibData", "calInfo", "observed", "minmaxt"))
clusterEvalQ(cl, library("osmose"))
clusterEvalQ(cl, library("calibrar"))

call = calibrate(calibData['paropt'], fn=objfn, method='default',
                 lower=calibData['parmin'], upper=calibData['parmax'],
                 phases=calibData['parphase'], control=control, replicates=2)

stopCluster(cl)
```

Danger: The way to run R in parallel depends on the calculation platform used!

Caution: The cluster must be closed after the `calibrate` function.

4.6 Creation of a `runModel` function

The next step is to create a `runModel` function, the purpose of which is to preprocess parameter files, run OSMOSE Java and postprocess the outputs.

The `runModel` function used to calibrate the [HLS+16] configuration is detailed below.

Danger: This `runModel.R` script is built for the reference configuration and must not be used to run other configurations. It must be adapted by the user.

4.6.1 Function arguments

This function should be adapted by the user. It must take at least two arguments:

- the array of calibrated parameters (param arguments)
- the names of the calibrated parameters (names arguments)

```
runModel = function(param, names, ...) {
```

4.6.2 Writes the calibrated parameters in the CSV file

The first step is to write the calibrated parameters in the calibration-parameters.csv file (that will be located in the master directory):

```
# set parameter names
names(param) = names

# writes the calibrated parameters into a CSV file
# following Osmose format. The parameters in this file
# will overwrite the Osmose parameter
write.table(param, file="calibration-parameters.csv", sep=";",
            col.names=FALSE, quote=FALSE)
```

Danger: The file format must be consistent with the expectations of OSMOSE Java.

4.6.3 Run OSMOSE Java

The next step is to run the OSMOSE Java program. This is done by using the `osmose::run_osmose` function:

```
# defines the user directory
outdir = "output"

cat("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ", getwd(), "\n")

# run Osmose Model
run_osmose(input="calib_config.csv",
```

Here, the `input` argument is the calibration configuration file used by OSMOSE Java. In this case, it is `calib_config.csv`, which includes the `calibration-parameters.csv` and the `osm_all-parameters.csv` (see [Table 4.1](#)).

4.6.4 Process OSMOSE Java outputs

The last step is to process OSMOSE Java outputs, so that they fit the observation time series (see [Section 4.2](#)).

The reading of OSMOSE Java outputs is achieved by using the `osmose::read_osmose` function.

```
verbose=TRUE, clean=TRUE)
```

Since biomass and yields observations are on yearly time-scales, the monthly simulated biomass and yields must be yearly integrated:

```
data = read_osmose(path=outdir)

# extract the biomass and yields variables (monthly data).
# expectes = TRUE to average over the replicates
osmose.biomass = get_var(data, "biomass", expected=TRUE)
osmose.thresholds = get_var(data, "biomass", expected=TRUE)
osmose.yields = get_var(data, "yield", expected=TRUE)

# define a year factor for yearly integration of biomass
# from monthly biomass
biomassDim = dim(osmose.biomass) # dims=(time, species, replic)
ntime = biomassDim[1] # nyears * 12
nspecies = biomassDim[2]
nyears = ntime / 12
years = factor(rep(1:nyears, each=12))

# Integration of monthly values into yearly values
# here, tapply is applied on dimension 1 (i.e. time)
```

Finally, the output must be converted into a list, whose elements' names must be consistent with the observation names (variable column in Table 4.4).

```
osmose.yields = apply(osmose.yields, 2, tapply, years, sum)

output = list(# Biomass
              OctopusVulgaris.biomass = param["q.sp0"]*osmose.biomass[,
↪ "OctopusVulgaris"],
              MelicertusKerathurus.biomass = param["q.sp1"]*osmose.biomass[,
↪ "MelicertusKerathurus"],
              MetapenaeusMonoceros.biomass = param["q.sp2"]*osmose.biomass[,
↪ "MetapenaeusMonoceros"],
              TrachurusTrachurus.biomass = param["q.sp3"]*osmose.biomass[,
↪ "TrachurusTrachurus"],
              SardinaPilchardus.biomass = param["q.sp4"]*osmose.biomass[,
↪ "SardinaPilchardus"],
              SardinellaAurita.biomass = param["q.sp5"]*osmose.biomass[,
↪ "SardinellaAurita"],
              EngraulisEncrasicolus.biomass = param["q.sp6"]*osmose.biomass[,
↪ "EngraulisEncrasicolus"],
              DiplodusAnnularis.biomass = param["q.sp7"]*osmose.biomass[,
↪ "DiplodusAnnularis"],
              MustelusMustelus.biomass = param["q.sp8"]*osmose.biomass[,
↪ "MustelusMustelus"],
              MerlucciusMerluccius.biomass = param["q.sp9"]*osmose.biomass[,
↪ "MerlucciusMerluccius"],
              PagellusErythrinus.biomass = param["q.sp10"]*osmose.biomass[,
↪ "PagellusErythrinus"],
              # Landings
              OctopusVulgaris.landings = osmose.yields[, "OctopusVulgaris"],
              MelicertusKerathurus.landings = osmose.yields[, "MelicertusKerathurus
↪"],
```

(continues on next page)

(continued from previous page)

```

↪",      MetapenaeusMonoceros.landings = osmose.yields[, "MetapenaeusMonoceros
TrachurusTrachurus.landings    = osmose.yields[, "TrachurusTrachurus"],
SardinaPilchardus.landings     = osmose.yields[, "SardinaPilchardus"],
SardinellaAurita.landings      = osmose.yields[, "SardinellaAurita"],
EngraulisEncrasicolus.landings = osmose.yields[, "EngraulisEncrasicolus
↪",
DiplodusAnnularis.landings     = osmose.yields[, "DiplodusAnnularis"],
MustelusMustelus.landings      = osmose.yields[, "MustelusMustelus"],
MerlucciusMerluccius.landings  = osmose.yields[, "MerlucciusMerluccius
↪",
PagellusErythrinus.landings    = osmose.yields[, "PagellusErythrinus"],
# Thresholds
OctopusVulgaris.thr            = osmose.thresholds[, "OctopusVulgaris"],
MelicertusKerathurus.thr       = osmose.thresholds[,
↪"MelicertusKerathurus"],
MetapenaeusMonoceros.thr       = osmose.thresholds[,
↪"MetapenaeusMonoceros"],
TrachurusTrachurus.thr         = osmose.thresholds[, "TrachurusTrachurus
↪",
SardinaPilchardus.thr          = osmose.thresholds[, "SardinaPilchardus
↪",
SardinellaAurita.thr           = osmose.thresholds[, "SardinellaAurita
↪",
EngraulisEncrasicolus.thr      = osmose.thresholds[,
↪"EngraulisEncrasicolus"],
DiplodusAnnularis.thr          = osmose.thresholds[, "DiplodusAnnularis
↪",
MustelusMustelus.thr           = osmose.thresholds[, "MustelusMustelus
↪",
MerlucciusMerluccius.thr       = osmose.thresholds[,
↪"MerlucciusMerluccius"],
PagellusErythrinus.thr         = osmose.thresholds[, "PagellusErythrinus
↪"]
)

```

4.7 Run the calibration

When everything is ready, run the calibration as follows:

```
R CMD BATCH calibrate.R
```


BIBLIOGRAPHY

MISCELLANEOUS

6.1 Datarmor use

6.1.1 Package loading

In order to use the `osmose` and `calibrar` R packages on Datarmor, the first step is to load the R module as follows:

```
module load R
module load nco
```

The following modules will be loaded:

```
Currently Loaded Modulefiles:
1) nco/4.7.1_conda           4) impi/2017.2.174
2) intel-cc-17/17.0.2.174   5) intel-cmkl-17/17.0.2.174
3) java/1.8.0               6) R/3.4.3-intel-17.0.2.174
```

Warning: The `nco` module must be loaded in order to use the R `ncdf4` library.

The second step is to define where the libraries are located. In order to avoid multiple copies, a possibility is to use the R libraries that have been built in Nicolas Barrier's home. This is done as follows:

```
# CSH users
setenv R_LIBS_USER /home1/datahome/nbarrier/libs/R/lib
```

```
# BASH/SH users
export R_LIBS_USER=/home1/datahome/nbarrier/libs/R/lib
```

To test whether the libraries are found, run R and types:

```
library("osmose")
library("calibrar")
```

6.1.2 Running parallel R programs

Running parallel R programs in Datarmor can be achieved in multiple ways (see the examples in */appli/services/examples/R/*).

Running on multiple nodes (MPI)

To run the calibration on multiple nodes, the calibration must be run by using the RMPISNOW program. The calibration is run by using the following PBS file:

```
#!/bin/csh
#PBS -q mpi_2
#PBS -l select=2:ncpus=28:mpiprocs=14:mem=125g
#PBS -l walltime=24:00:00

cd $PBS_O_WORKDIR
echo $HOST
pbsnodes $HOST

# recovering the number of MPI processes (here, 2 * 14 = 28)
setenv mpiproc `cat $PBS_NODEFILE |wc -l`

# load the R libraries
source /usr/share/Modules/3.2.10/init/csh
module load R
module load nco

# set the path of the osmose/calibrar libraries
setenv R_LIBS /home1/datahome/nbarrier/libs/R/lib

# Run R in parallel mode.
time mpiexec -np $mpiproc /appli/R/3.3.2-intel-cc-17.0.2.174/lib64/R/library/snow/
↳RMPISNOW --no-save -q < calibrate_MPI.R >& ea.out
```

Hint: It is possible to use BASH instead of CSH. However, it is highly advised to use CSH, since it is the default Datarmor shell.

When using RMPISNOW, the parallel library that must be used is the doSNOW package. As a consequence, the calibration script must be modified as follows:

```
# Need to load the doSNOW package, which is
# installed in Nicolas Barrier's home
require("doSNOW")

# call the RMPI/Snow make cluster (note here that there are no arguments!)
cl <- makeCluster()

# call the registerDoSNOW function instead of the registerDoParallel
registerDoSNOW(cl)

# send the variables and loaded libraries defined in the above to the nodes
```

(continues on next page)

(continued from previous page)

```

clusterExport(cl, c("objfn", "calibData", "calInfo", "observed", "minmaxt"))
clusterEvalQ(cl, library("osmose"))
clusterEvalQ(cl, library("calibrar"))

# run the calibration
cal1 = calibrate(calibData['paropt'], fn=objfn, method='default',
                lower=calibData['parmin'], upper=calibData['parmax'],
                phases=calibData['parphase'], control=control, replicates=1)

# stop the cluster
stopCluster(cl)

```

The main differences between this R script and the one described in [Section 4.5.8](#) are:

- `require("doSNOW")` instead of `require("parallel")`
- No arguments in the `makeCluster` function
- `registerDoSNOW` instead of `registerDoParallel`

Running on a single node

To run the calibration in parallel on a single node (for instance on a Shared Memory machine), the `doParallel` library is used. In this case, the PBS file is as follows:

```

#!/bin/csh
#PBS -q omp
#PBS -l select=1:ncpus=28:mem=120g
#PBS -l walltime=24:00:00

cd $PBS_O_WORKDIR
echo $HOST
pbsnodes $HOST

# load the R libraries
source /usr/share/Modules/3.2.10/init/csh
module load R
module load nco

# set the path of the osmose/calibrar libraries
setenv R_LIBS /home1/datahome/nbarrier/libs/R/lib

# run R in parallel mode
time R --vanilla < calibrate_OMP.R >& ea.out

```

The `calibrate.R` script must be modified as follows:

```

# With OMP, you need to load the doParallel library
require("doParallel")

# Initialisation of the cluster.
# BE SURE THAT THE NUMBER OF CORE HERE IS
# CONSISTENT WITH THE NUMBER OF YOUR PBS FILE

```

(continues on next page)

(continued from previous page)

```

cl <- makeCluster(control$nCores)

# register the doParallel so that for each is activated
registerDoParallel (cl)

# send the variables and loaded libraries defined in the above to the nodes
clusterExport(cl, c("objfn", "calibData", "calInfo", "observed", "minmaxt"))
clusterEvalQ(cl, library("osmose"))
clusterEvalQ(cl, library("calibrar"))

# run the calibration
call = calibrate(calibData['paropt'], fn=objfn, method='default',
                lower=calibData['parmin'], upper=calibData['parmax'],
                phases=calibData['parphase'], control=control, replicates=1)

# stop the cluster
stopCluster(cl)

```

The main difference between this R script and the one described in [Section 4.5.8](#) is:

- `require("doParallel")` instead of `require("parallel")`

Danger: The number of cores defined in the .pbs file (ncpus) must be consistent with the value in the `control$nCores` parameter.

Running on a single core (sequential)

To run R in parallel on a single core (i.e. in sequential), the PBS file must be as follows:

```

#!/bin/csh
#PBS -l walltime=24:00:00
#PBS -l mem=1g

cd $PBS_O_WORKDIR
echo $HOST
pbsnodes $HOST

# load the R libraries
source /usr/share/Modules/3.2.10/init/csh
module load R
module load nco

# set the path of the osmose/calibrar libraries
setenv R_LIBS /home1/datahome/nbarrier/libs/R/lib

# Run R in parallel mode.
time R --vanilla < calibrate_SEQ.R >& ea.out

```

In this case, no parallel libraries need to be loaded. Hence, the script described in [Section 4.5](#) can be used without any modifications.

PARAMETER INDEX

BIBLIOGRAPHY

- [FOT+17] Caihong Fu, Norm Olsen, Nathan Taylor, Arnaud Grüss, Sonia Batten, Huizhu Liu, Philippe Verley, and Yunne-Jai Shin. Spatial and temporal dynamics of predator-prey species interactions off western Canada. *ICES Journal of Marine Science*, 74(8):2107–2119, 05 2017. URL: <https://dx.doi.org/10.1093/icesjms/fsx056>, arXiv:<http://oup.prod.sis.lan/icesjms/article-pdf/74/8/2107/20143985/fsx056.pdf>, doi:10.1093/icesjms/fsx056.
- [FPS+13] Caihong Fu, R Ian Perry, Yunne-Jai Shin, Jake Schweigert, and Huizhu Liu. An ecosystem modelling framework for incorporating climate regime shifts into fisheries management. *Progress in oceanography*, 115:53–64, 2013.
- [GrussSC+15] Arnaud Grüss, Michael J Schirripa, David Chagaris, Michael Drexler, James Simons, Philippe Verley, Yunne-Jai Shin, Mandy Karnauskas, Ricardo Oliveros-Ramos, and Cameron H Ainsworth. Evaluation of the trophic structure of the west florida shelf in the 2000s using the ecosystem model osmose. *Journal of Marine Systems*, 144:30–47, 2015.
- [HLS+16] Ghassen Halouani, Frida Ben Rais Lasram, Yunne-Jai Shin, Laure Velez, Philippe Verley, Tarek Hattab, Ricardo Oliveros-Ramos, Frédéric Diaz, Frédéric Ménard, Melika Baklouti, Arnaud Guyennon, Mohamed Salah Romdhane, and François Le Loch. Modelling food web structure using an end-to-end approach in the coastal ecosystem of the Gulf of Gabes (Tunisia). *Ecological Modelling*, 339(Supplement C):45–57, 2016. URL: <http://www.sciencedirect.com/science/article/pii/S0304380016303118>, doi:<https://doi.org/10.1016/j.ecolmodel.2016.08.008>.
- [OR14] Ricardo Oliveros Ramos. *End-to-end modelling for an ecosystem approach to fisheries in the Northern Humboldt Current Ecosystem*. PhD thesis, University of Montpellier, France, 2014.
- [ORS17] Ricardo Oliveros-Ramos and Yunne-Jai Shin. calibrar: an R package for fitting complex ecological models. ????, ???:???, ??? 2017.
- [ORVES17] Ricardo Oliveros-Ramos, Philippe Verley, Vincent Echevin, and Yunne-Jai Shin. A sequential approach to calibrate ecosystem models with multiple time series data. *PROGRESS IN OCEANOGRAPHY*, 151:227–244, FEB 2017. doi:{10.1016/j.pocean.2017.01.002}.
- [SC01] YJ Shin and P Cury. Exploring fish community dynamics through size-dependent trophic interactions using a spatialized individual-based model. *AQUATIC LIVING RESOURCES*, 14(2):65–80, MAR-APR 2001. doi:{10.1016/S0990-7440(01)01106-8}.
- [SC04] YJ Shin and P Cury. Using an individual-based model of fish assemblages to study the response of size spectra to changes in fishing. *CANADIAN JOURNAL OF FISHERIES AND AQUATIC SCIENCES*, 61(3):414–431, MAR 2004. doi:{10.1139/F03-154}.
- [TSSC06] M Travers, YJ Shin, L Shannon, and P Cury. Simulating and testing the sensitivity of ecosystem-based indicators to fishing in the southern Benguela ecosystem. *CANADIAN JOURNAL OF FISHERIES AND AQUATIC SCIENCES*, 63(4):943–956, APR 2006. doi:{10.1139/1706-003}.

- [TSJ+09] M. Travers, Y.-J. Shin, S. Jennings, E. Machu, J.A. Huggett, J.G. Field, and P.M. Cury. Two-way coupling versus one-way forcing of plankton and fish models to predict ecosystem changes in the benguela. *Ecological Modelling*, 220(21):3089 – 3099, 2009. Selected Papers from the Sixth European Conference on Ecological Modelling - ECEM '07, on Challenges for ecological modelling in a changing world: Global Changes, Sustainability and Ecosystem Based Management, November 27-30, 2007, Trieste, Italy. URL: <http://www.sciencedirect.com/science/article/pii/S0304380009005766>, doi:<https://doi.org/10.1016/j.ecolmodel.2009.08.016>.
- [TTSF14] M Travers-Trolet, Y-J Shin, and JG Field. An end-to-end coupled model roms-n2p2z2d2-osmose of the southern benguela foodweb: parameterisation, calibration and pattern-oriented validation. *African Journal of Marine Science*, 36(1):11–29, 2014. URL: <https://doi.org/10.2989/1814232X.2014.883326>, arXiv:<https://doi.org/10.2989/1814232X.2014.883326>, doi:10.2989/1814232X.2014.883326.

INDEX

B

biomass.byDt.bySize.file.bkg#, 16

F

fisheries.catchability.file, 51
fisheries.catchability.file.cat#, 51
fisheries.catchability.finalYear.cat#, 51
fisheries.catchability.initialYear.cat#, 51
fisheries.catchability.steps.cat#, 51
fisheries.catchability.years.cat#, 51
fisheries.check.enabled, 42
fisheries.discards.file, 52
fisheries.discards.file.dis#, 52
fisheries.discards.finalYear.dis#, 52
fisheries.discards.initialYear.dis#, 52
fisheries.discards.steps.dis#, 52
fisheries.discards.years.dis#, 52
fisheries.period.number.fsh#, 43
fisheries.period.start.fsh#, 43
fisheries.rate.base.fsh#, 43
fisheries.rate.base.log.enabled.fsh#, 43
fisheries.rate.base.shift.fsh#, 43
fisheries.rate.byperiod.fsh#, 43
fisheries.seasonality.file.fsh#, 43
fisheries.seasonality.fsh#, 43
fisheries.selectivity.a50.file.fsh#, 50
fisheries.selectivity.a50.fsh#, 50
fisheries.selectivity.a50.shift.fsh#, 50
fisheries.selectivity.l50.file.fsh#, 50
fisheries.selectivity.l50.fsh#, 50
fisheries.selectivity.l50.shift.fsh#, 50
fisheries.selectivity.l75.file.fsh#, 50
fisheries.selectivity.l75.fsh#, 50
fisheries.selectivity.l75.shift.fsh#, 50
fisheries.selectivity.tiny.fsh#, 50
fisheries.selectivity.type.file.fsh#, 50
fisheries.selectivity.type.fsh#, 50
fisheries.selectivity.type.shift.fsh#, 50

G

grid.lowright.lat, 13
grid.lowright.lon, 13

grid.mask.file, 13
grid.ncolumn, 13
grid.netcdf.file, 13, 18
grid.nline, 13
grid.stride, 13, 18
grid.upleft.lat, 13
grid.upleft.lon, 13
grid.var.lat, 13
grid.var.lon, 13
grid.var.mask, 13
growth.exponential.ke.sp#, 25
growth.exponential.lstart.sp#, 25
growth.exponential.thr.age.sp#, 25
growth.gompertz.kg.sp#, 25
growth.gompertz.linf.sp#, 25
growth.gompertz.tg.sp#, 25
growth.gompertz.thr.age.sp#, 25
growth.java.classname.sp#, 24

L

l1l.integration.depth, 18
l1l.netcdf.dim.ntime, 18
l1l.netcdf.file, 19
l1l.netcdf.file.plk#, 19
l1l.netcdf.file.t#, 18
l1l.netcdf.grid.file, 18
l1l.netcdf.var.bathy, 18
l1l.netcdf.var.csr, 18
l1l.netcdf.var.hc, 18
l1l.netcdf.var.lat, 18
l1l.netcdf.var.lon, 18
l1l.netcdf.var.plankton.plk#, 18
l1l.netcdf.var.zlevel, 18
l1l.nstep, 18, 19

M

mortality.algorithm, 29
mortality.fishing.catches.byDt.byAge.file.sp#, 35
mortality.fishing.catches.byDt.bySize.file.sp#, 35
mortality.fishing.catches.byYear.file.sp#, 35

mortality.fishing.catches.sp#, 35
mortality.fishing.rate.byDt.byAge.file.sp#, 35
mortality.fishing.rate.byDt.bySize.file.sp#, 35
mortality.fishing.rate.byYear.file.sp#, 35
mortality.fishing.rate.sp#, 35
mortality.fishing.recruitment.age.sp#, 34
mortality.fishing.recruitment.size.sp#, 34
mortality.fishing.season.distrib.file.sp#, 35
mortality.fishing.type, 34
mortality.out.rate.sp#, 34
mortality.starvation.rate.max.sp#, 34
mortality.subdt, 29
movement.bkgSpecies.class.map#, 16
movement.bkgSpecies.season.map#, 16
movement.bkgSpecies.species.map#, 16
movement.bkgSpecies.year.max.map#, 16
movement.bkgSpecies.year.min.map#, 16
mpa.end.year.mpa#, 35
mpa.file.mpa#, 35
mpa.start.year.mpa#, 35

P

plankton.accessibility2fish.plk#, 15
plankton.biomass.total.plk#, 19
plankton.conversion2tons.plk#, 18
plankton.multiplier.plk#, 19
plankton.name.plk#, 15
plankton.size.max.plk#, 15
plankton.size.min.plk#, 15
plankton.TL.plk#, 15
predation.accessibility.file, 32
predation.accessibility.file.acc#, 32
predation.accessibility.finalYear.acc#, 32
predation.accessibility.initialYear.acc#, 32
predation.accessibility.stage.structure, 32
predation.accessibility.stage.threshold.bkg#, 16
predation.accessibility.stage.threshold.sp#, 32
predation.accessibility.steps.acc#, 32
predation.accessibility.years.acc#, 32
predation.efficiency.critical.bkg#, 16
predation.efficiency.critical.sp#, 17, 24, 34
predation.ingestion.rate.max.bkg#, 16
predation.ingestion.rate.max.sp#, 17, 33
predation.predPrey.sizeRatio.max.bkg#, 16
predation.predPrey.sizeRatio.max.sp#, 17, 30
predation.predPrey.sizeRatio.min.bkg#, 16
predation.predPrey.sizeRatio.min.sp#, 17, 30
predation.predPrey.stage.structure, 30
predation.predPrey.stage.threshold.bkg#, 16
predation.predPrey.stage.threshold.sp#, 17, 30

S

species.accessibility2fish.sp#, 15
species.age.sp#, 17
species.biomass.total.sp#, 17, 20
species.delta.lmax.factor.sp#, 24
species.file.caching.sp#, 17, 20
species.file.sp#, 17, 20
species.k.sp#, 25
species.length.sp#, 17
species.length2weight.allometric.power.bkg#, 16
species.length2weight.allometric.power.sp#, 17
species.length2weight.condition.factor.bkg#, 16
species.length2weight.condition.factor.sp#, 17
species.linf.sp#, 25
species.name.bkg#, 16
species.name.sp#, 15, 17
species.nclass.sp#, 17
species.size.max.sp#, 15
species.size.min.sp#, 15
species.size.proportion.sp#, 17
species.t0.sp#, 25
species.TL.sp#, 15
species.trophiclevel.bkg#, 16
species.trophiclevel.sp#, 17
species.type.sp#, 15
species.vonbertalanffy.threshold.age.sp#, 25
stochastic.mortality.seed, 29