Osmose User Guide

Nicolas Barrier Philippe Verley Ricardo Oliveros-Ramos
Bruno Ernande Wencheng Lau-Medrano Criscely Luján
Alaia Morell Morgane Travers-Trolet Laure Velez
Yunne-Jai Shin

Table of contents

1 Introduction

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 characterised by their size, weight, age, taxonomy and geographic location (2D model). Each school is subject to different processes of the fish life cycle (growth, explicit predation, natural and starvation mortality, reproduction and migration) in addition to fishing mortality, which is different for each species and structured by age/size, space and season.

The model requires basic biological parameters for growth and reproduction processes, which are often available for a wide range of species, and which can be found, for example, in FishBase. It also needs to be forced by spatial distribution maps for each species, by age/size/stage and by season depending on data availability. Osmose provides a variety of size and species based ecological indicators can be produced and compared with 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)
- 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 with various hydrodynamic and biogeochemical models, allowing for the construction of end-to-end models of marine ecosystems that explicitely account for the combined effects of climate and fishing on fish dynamics.

2 Download requirements

In this section you will find the list of software and libraries that are required to use the full capabilities of the Osmose model.

2.1 Download Java

Since the Osmose numerical core is coded in Java, let us clarify some of the acronyms related to the Java technologies.

- `JVM`: Java Virtual Machine. This is a set of software programs that interprets the Java byte code.
- `JRE`: Java Runtime Environment. This is a kit distributed by Sun for running Java programs. A `JRE` provides a `JVM` and some basic Java libraries. **A JRE is required to run Osmose**.
- `JDK` or `SDK`: Java (or Software) Development Kit used by 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 required if the precompiled Java binaries are not used**

We strongly recommend the OpenJDK suite, available from https://jdk.java.net/ :

- Download the archive file (`.tar.gz` for `.zip` extension).
- Extract the files to a dedicated folder.
- Set the `JAVA_HOME` environment variable to the location of the Java folder.

Tip

To find out how to set environment variables in Windows, click here. For Linux/Mac Os users, click here

❗ Important

The OpenJDK version must be at least 8

2.2 Installing NetCDF library

As Osmose makes extensive use of NetCDF files, the NetCDF4 library must be installed.

2.2.1 Mac Os X

To install the library on a Mac OS system, open a terminal and type:

```
sudo port install netcdf4
```

2.2.2 Linux

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

```
sudo apt install netcdf4
```

2.2.3 Windows

Download and install the NetCDF library from the Unidata website. Select the `netCDFX.Y.Z-NC4-DAP-64.exe` install.

> ⚠ Warning
>
> Ensure that the NetCDF binaries are added to the `PATH` environment variable.

2.3 Download R

The Osmose Java core is embedded in an Osmose R package, which allows pre-processing, calibration, execution and post-processing of Osmose configurations. It is strongly recommended to install R to take advantage of the features provided by the R package.

Download instructions for R are available for Windows, Linux and Mac Os X.

It is also recommended that you install the RStudio GUI (https://rstudio.com/).

> Tip
>
> If you do not intend to use the R package, you can skip this part.

2.4 Installing R libraries

To install the Osmose package, R libraries are required (see the `DESCRIPTION` file). These libraries can be installed by typing from an R console:

```r
install.packages(c("calibrar", "knitr", "ncdf4", "rlist",
                   "rmarkdown", "stringr", "fields",
                   "R.utils"))
```

3 Installing Osmose

The Osmose model is provided as a combination of a Java numerical core (referred to as Osmose Java), linked to an Osmose R package (referred to as Osmose R). The code is available on a GitHub repository.

It is strongly recommended to install Osmose from R, in order to have access to all the features provided by the package. There are several ways to install Osmose from the Git repository.

3.1 Install using devtools

It is possible to install the Osmose package using the `install_github` function of the `devtools` package. First, open an R session (by typing `R` from a terminal or CMD prompt). In the R console, type:

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

This will download the latest commit from the `master` branch and use it to compile the Osmose package.

> **Note**
>
> The `devtools` package is required

3.2 Install from RStudio

To install Osmose using RStudio, click on the `File --> New Project` menu and open the `Version Control --> Git` menu.

Set the package URL to https://github.com/osmose-model/osmose.

Once the project is open, click on the `Build & Reload` button to install the package.

Note

The `devtools` package need to be installed.

3.3 Java executable

When installing the Osmose R package, no java executable is provided. The latter is automatically downloaded by the R package when the `run_osmose` package function is called.

The location where the executables are to be downloaded must be specified in a `.Renviron` file (see this article for a description of `.Renviron` files).

Create or edit the `.Renviron` file and set the `OSMOSE_DIR` variable to your destination folder. For example:

```
OSMOSE_DIR=/home/barrier/Work/codes/osmose-executables
```

If no `.Renviron` file exists or if the `OSMOSE_DIR` variable does not exist, the Java executables are downloaded to a temporary folder anytime the `run_osmose` function is called.

Note that the Java executables are available in the Releases section of the Github repository. The Java executables are named `osmose-X.Y.Z-jar-with-dependencies.jar`, where `X.Y.Z` is the Osmose version. These Java executables contain all the external Java libraries required by Osmose (NetCDF, etc.).

3.4 Install from source files

To install Osmose from source files, you first need to download the Java compilation tool Maven.

Next, download the code. You can either download it from the archive file or by using the `git clone` command, as follows:

```
git clone https://github.com/osmose-model/osmose.git
```

Tip

It is strongly recommended that your Git installation was built with Large File Storage (LFS) support. This is the case on Windows, not on Linux or Mac Os X. Visit LFS website for more informations

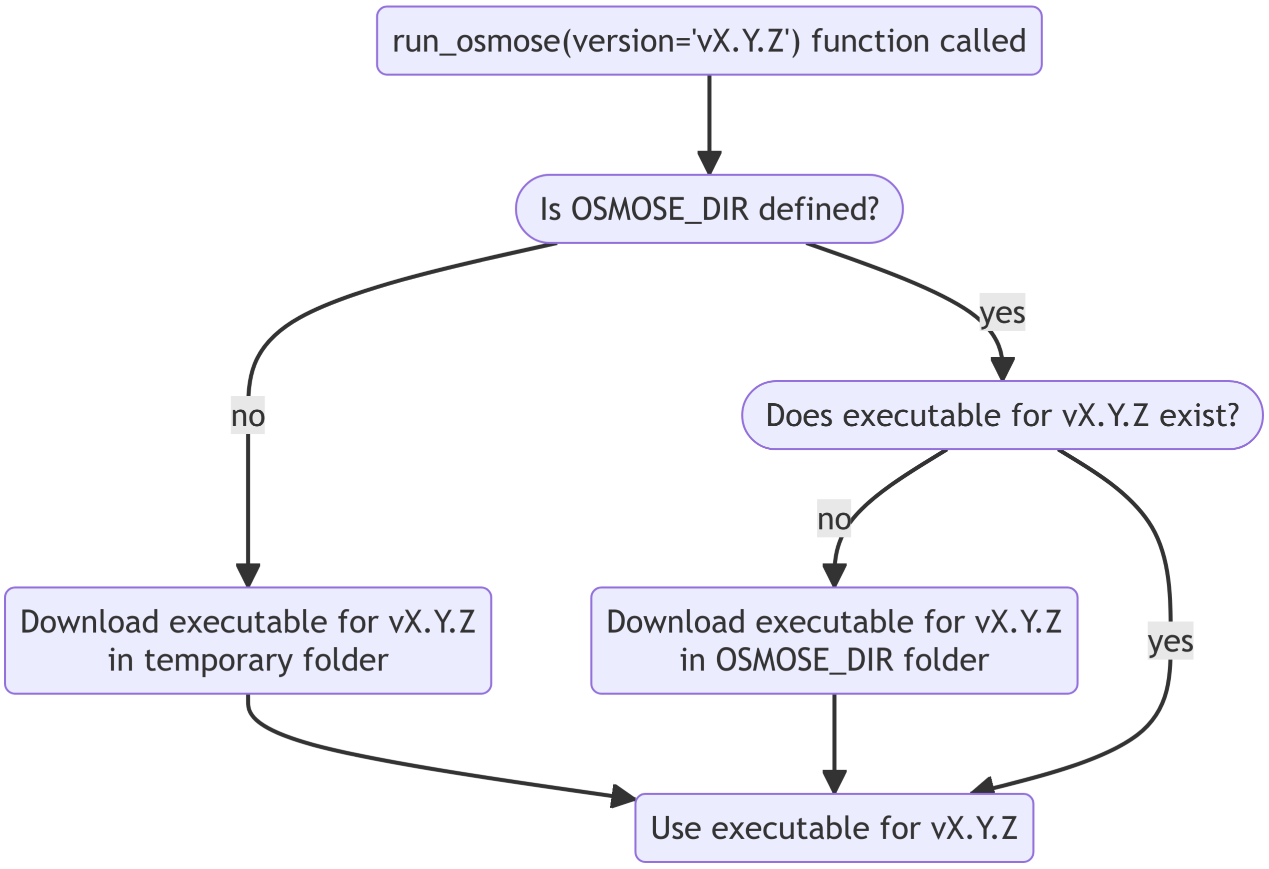

run_osmose(version='vX.Y.Z') function called
Is OSMOSE_DIR defined?
yes
no
Does executable for vX.Y.Z exist?
no
yes
Download executable for vX.Y.Z in temporary folder
Download executable for vX.Y.Z in OSMOSE_DIR folder
Use executable for vX.Y.Z

Figure 3.1: Process for downloading Osmose Java executable when running Osmose from R

When the code has been downloaded, go into the `osmose` folder and compile the Java code as follows:

```
mvn package
```

Tip

The Java compilation can also be performed using any Java IDE, such as NetBeana or VSCode

This step will generate the `osmose-X.Y.Z-jar-with-dependencies.jar` executable and put it into the `inst` folder, which will be used in the R package install process.

Next, install the package using the following command:

```
R CMD INSTALL .
```

Tip

This step can be achieved by using RStudio. After opening the `osmose` folder as a new project, click on the `Build & Reload` button to install the package.

Note

The `devtools` package need to be installed.

During this operation, the Java executable, which was stored in the `inst` folder, will be copied into the R libs folder. **Therefore, the R package must be reinstalled any time the Java core is recompiled**. These steps are summarized below.

Tip

We strongly recommend that you favour the `git clone` method. Updating the code can be simply done by pulling changes from the remote folder using `git pull`

Important

Any time the Java sources are updated, both the Java and the R must be recompiled as shown in Figure 3.2

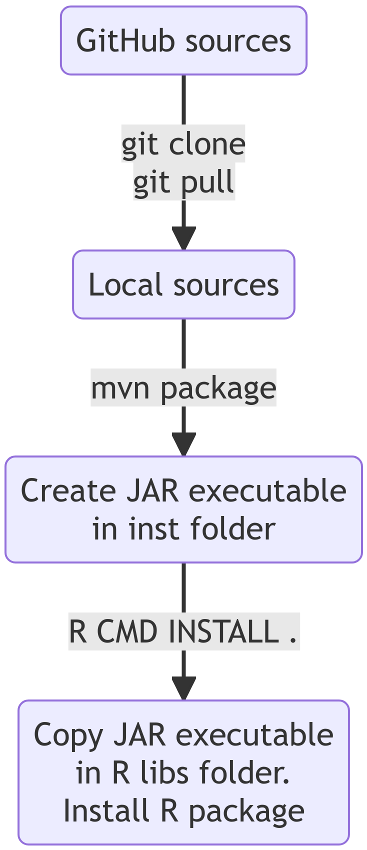

GitHub sources
git clone
git pull
Local sources
mvn package
Create JAR executable in inst folder
R CMD INSTALL .
Copy JAR executable in R libs folder. Install R package

Figure 3.2: Process for installing osmose from source files

3.5 For developers

The development version of Osmose is available on a private GitHub repository: https://github.com/osmose-model/osmose-private.git.

To be able to clone the private repository, you will need to authentify to GitHub (see here for details).

4 Datarmor install

This section describes the easiest way to install Osmose on the Datarmor HPC. When running a job on Datarmor, you do not have access to the Internet. Therefore, the Java executables cannot be downloaded as discussed in Section 3.3. As a consequence, both the Java core and the R package must be compiled manually following Section 3.4.

In the following, we describe the steps to achieve this install on Datarmor.

4.1 Loading the modules

From the Terminal, first load the necessary modules:

```
module load R/3.6.3-intel-cc-17.0.2.174
module load java/openjdk-16.0.2
module load nco
```

! Important

To use parallel R features, this specific R module is required

i Note

The `nco` module is needed to have access to the NetCDF library

4.2 Set-up environment variables

Edit the `~/.Renviron` file and define the `R_LIBS_USER` environment variables as follows:

```
R_LIBS_USER=/home1/datawork/nbarrier/libs/R/lib
```

It specifies the location where the R libraries will be insalled.

Note

The definition of the `OSMOSE_DIR` variable is not required since the Java sources will be generated by the compilation process.

4.3 Setting up conda

To have access to Git and Maven, the first step is to enable the use of Conda, through which these tools are available.

First, create a `~/.condarc` file, which contains the following lines:

```
envs_dirs:
  - ${DATAWORK}/anaconda3-envs
  - /home1/datahome/nbarrier/softwares/anaconda3-envs
  - /appli/conda-env
  - /appli/conda-env/2.7
  - /appli/conda-env/3.6
channels:
  - conda-forge
  - defaults
```

When this is done, if using CSH, type:

```
source /appli/anaconda/latest/etc/profile.d/conda.csh
```

If using BASH, type:

```
. /appli/anaconda/latest/etc/profile.d/conda.sh
```

Tip

To make Conda permanently available, you can add the line above commands to your `~/.cshrc` or `~/.bashrc` files. file

4.4 Cloning the code

Once Conda is set up, activate the Git environment by typing:

```
conda activate git
```

This will give you access to a `git` executable configured with the LFS support.

Finally, type the following:

```
git clone https://github.com/osmose-model/osmose.git
```

4.5 Compiling the code

To compile the Osmose Java code on Datarmor, you will need to use the `maven` conda environment. To activate it:

```
conda activate maven
```

Then, go in the `osmose` folder and type:

```
mvn package
```

The Java executable file is created in the `inst/java/` folder of the `osmose` directory.

4.6 Installing the package

Once the Java code is compiled, you can install the Osmose R package.

First, install the required libraries as described in Section 2.4.

Then, install the Osmose R package from the `osmose` folder:

```
R CMD INSTALL .
```

5 Running Osmose demo

Now that the R package is installed, let's try to run the Osmose demo configuration.

First, launch an R console and load the Osmose library:

```r
library(osmose)
ls("package:osmose")
```

```
 [1] "cacheManager"           "get_var"
 [3] "getVar"                 "initialize_osmose"
 [5] "osmose_calib_demo"      "osmose_demo"
 [7] "osmose_grid"            "osmose2R"
 [9] "read_osmose"            "readOsmoseConfiguration"
[11] "report"                 "run_osmose"
[13] "runOsmose"              "update_maps"
[15] "update_osmose"          "write_osmose"
[17] "write.osmose"
```

Now, call the `osmose_demo` function, which allows to copy the reference configuration into your working directory:

```r
demo = osmose_demo(path = "original-demo", config="eec_4.3.0")
demo
```

```
$config_file
[1] "original-demo/eec_4.3.0/eec_all-parameters.csv"

$output_dir
[1] "original-demo/eec_4.3.0/output-PAPIER-trophic"

$extra_args
[1] ""
```

The demo object contains the location of the configuration file and extra arguments.

If some parameters provided in the default configuration must be overwritten, an extra_args argument can be provided. It must have the following format:

```
extra_args = "-Psimulation.time.nyear=15 -Poutput.recordfrequency.ndt=1"
```

Here, the simulation.time.nyear and output.recordfrequency.ndt have been modified. The updated demo configuration can then be created follows:

```
demo2 = osmose_demo(path = "modified-demo", config="eec_4.3.0", extra_args=extra_args)
demo2
```

```
$config_file
[1] "modified-demo/eec_4.3.0/eec_all-parameters.csv"

$output_dir
[1] "modified-demo/eec_4.3.0/output-PAPIER-trophic"

$extra_args
[1] "-Psimulation.time.nyear=15 -Poutput.recordfrequency.ndt=1"
```

Now, you can run the Osmose model as follows:

```
run_osmose(demo$config_file, parameters=demo$extra_args, force=TRUE)
```

```
This is OSMOSE version 4.3.3

Running: 'java'  -jar '/home/BARRIER/Work/osmose/osmose-download-r/osmose-4.3.3-jar-with-dependenc
```

The first argument is the Osmose configuration file, the parameters argument allows to overwrite some of the original parameters. The force argument allows to run the configuration even if the configuration version number is inconsistent with the Osmose version.

> ⚠ Warning
>
> Use the force argument with caution!

If you installed Osmose from sources, as discussed in Section 3.4, and that you want to use the Jar file compiled with Maven, set the version argument to NULL:

```r
run_osmose(demo$config_file, parameters=demo$extra_args, force=TRUE, version=NULL)
```

```
This is OSMOSE version 4.3.3
```

```
Running: 'java'  -jar '/home/BARRIER/Libs/R/osmose/java/osmose_4.3.3-jar-with-dependencies.jar' or
```

While Osmose is running, an `osmose.log` file is written in the working directory. This file gives you some information about the simulation. If it has finished properly, you should see:

```
osmose[info] Simulation 0 completed (time ellapsed: 123 seconds)
osmose[info] *********************************************
osmose[info] OSMOSE Model copyright © IRD
osmose[info] *********************************************
```

Now that the simulation has been properly completed, you can read the Osmose output files by using the `read_osmose` function:

```r
output = read_osmose(demo$output_dir)
output
```

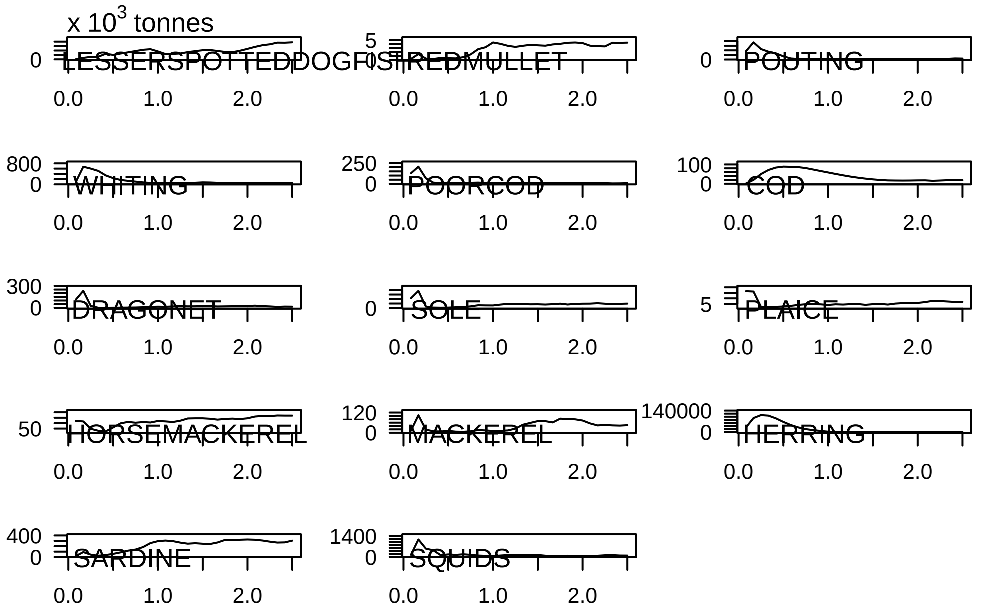
```
OSMOSE v.4
Model 'eec'

14 species modeled (1 simulations):
    [sp0] lesserSpottedDogfish
    [sp1] redMullet
    [sp2] pouting
    [sp3] whiting
    [sp4] poorCod
    [sp5] cod
    [sp6] dragonet
    [sp7] sole
    [sp8] plaice
    [sp9] horseMackerel
    [sp10] mackerel
    [sp11] herring
    [sp12] sardine
    [sp13] squids

Available fields:
```

```
"model"                      "sizeMature (*)"
"species"                    "ageMature (*)"
"biomass"                    "ingestion (*)"
"abundance"                  "ingestionTot (*)"
"mortality"                  "maintenance (*)"
"meanTL"                     "meanEnet (*)"
"meanTLCatch"                "sizeInf (*)"
"biomassByTL"                "kappa (*)"
"predatorPressure"           "abundAge1 (*)"
"predPreyIni"                "ingestByAge (*)"
"dietMatrix (*)"             "ingestBySize (*)"
"meanSize (*)"               "kappaByAge (*)"
"meanSizeCatch (*)"          "kappaBySize (*)"
"SizeSpectrum (*)"           "enetByAge (*)"
"abundanceBySize"            "enetBySize (*)"
"biomassBySize"              "maintenanceByAge (*)"
"meanTLBySize"               "maintenanceBySize (*)"
"mortalityBySize (*)"        "meanWeightByAge (*)"
"dietMatrixBySize (*)"       "meanWeightBySize (*)"
"predatorPressureBySize (*)" "meanWeightByWeight (*)"
"abundanceByAge (*)"         "yieldByWeight (*)"
"biomassByAge"               "yieldNByWeight (*)"
"meanSizeByAge"              "abundanceByWeight (*)"
"meanTLByAge (*)"            "biomassByWeight (*)"
"mortalityByAge (*)"         "yield.fishery0"
"dietMatrixByAge (*)"        "yield.fishery1"
"predatorPressureByAge (*)"  "yield.fishery2"
"abundanceByTL (*)"          "yield.fishery3"
"yieldByFishery"             "yield.fishery4"
"yield"                      "yield.fishery5"
"yieldN"                     "yield.fishery6"
"yieldBySize (*)"            "yield.fishery7"
"yieldNBySize (*)"           "yield.fishery8"
"yieldByAge (*)"             "yield.fishery9"
"yieldNByAge (*)"            "yield.fishery10"
"discards"                   "yield.fishery11"
"surveyBiomass (*)"          "yield.fishery12"
"surveyAbundance (*)"        "yield.fishery13"
"surveyYield (*)"            "config"
```

```
(*) Empty fields.
```

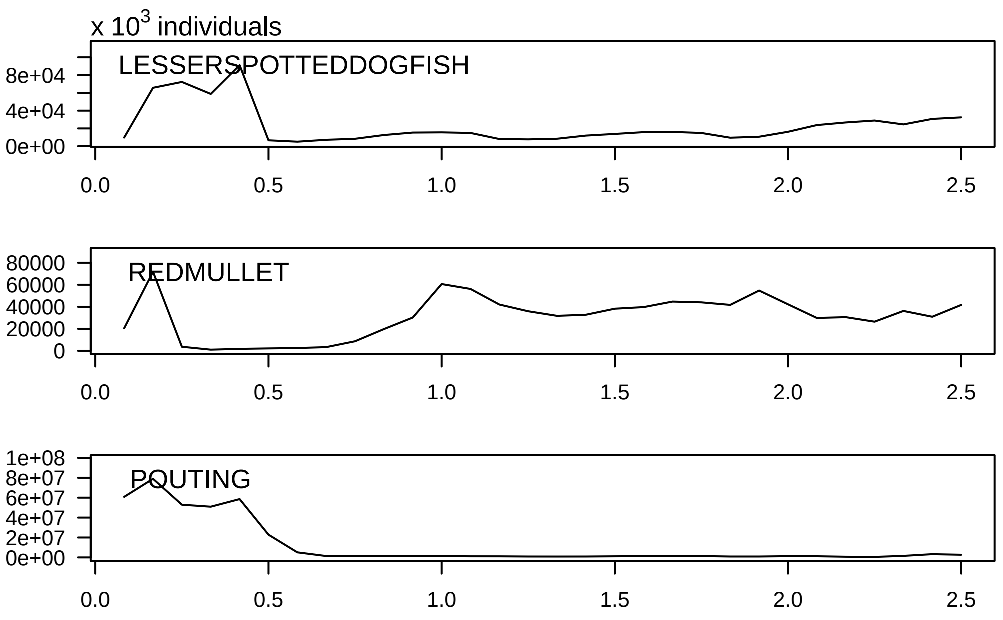
You can now plot some of the available fields using the `plot` function. For example, to plot the biomass for all the species:

```
plot(output, what="biomass")
```



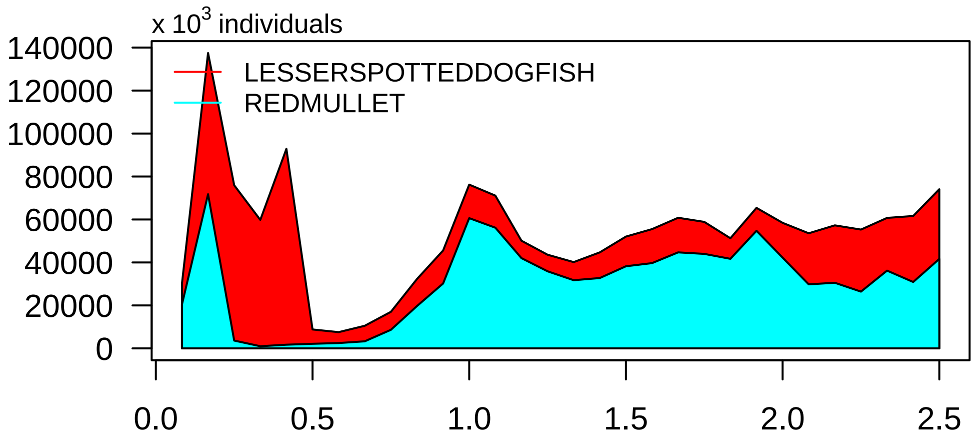
To plot the abundance of a single species:

```
plot(output, what="abundance", species=c(0, 1, 2))
```



You can also change the type of the plot as follows:

```
plot(output, what="abundance", species=c(0, 1), type=3)
```



6 Input data

This section described the input files that are required to run the Osmose model.

6.1 Configuration files

Running Osmose requires specifying the path to an Osmose configuration file.

An Osmose configuration file is a text based file. The name of the file doesn'nt matter, but we recommend that you avoid any special characters and spaces in the name of the file as this may cause 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 passed as an argument. The main configuration file can contain comments, empty lines and parameters. Osmose scans each line of the main configuration file looking for parameters and automatically discards:

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

> Tip
>
> 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**.

6.1.1 Key

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.`* have a special meaning for the configuration manager. It means that the value of this parameter is the path to another Osmose configuration file and the parameters in that file are to be loaded into the current configuration. This way, instead of having one big configuration file with all the parameters, it is possible to split the parameters into as many files as the user wants.

This process works recursively: a file can contain one or several `osmose.configuration` parameters that point to embedded configuration files, which may also contain one or several `osmose.configuration`parameters, 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).

⚠ Relative path

Paths parameters are always defined relative to the configuration file in which they are defined

6.1.2 Separators

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 (although it is advisable to be consistent and use the same one). The configuration manager will work out what the delimiter for each parameter.

6.1.3 Value

The value can be any sequence of characters (even empty). The configuration manager does not attempt to interpret the value when loading the configuration files. It simply stores it as a string object. A value can be passed to the configuration manager as:

- a string
- an integer
- a float

- a double
- a boolean
- an array of strings
- an array of integers
- an array of floats
- an array of doubles
- a resolved path

An array of values is a sequence of values with a separator in between. 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`.

6.1.4 Decimal separator

Osmose is quite flexible in terms of separators for the configuration files (automatically detected), the CSV output files (user-defined by parameter `output.csv.separator`) and the CSV input files (automatically detected). 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.

6.2 CSV input file separator

Many Osmose parameters are paths to CSV file, for instance:

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

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.

6.2.1 Spatial maps

The spatial dynamics of Osmose (species distribution, fishing dynamics, etc.) can be configured using CSV files. Each line of the CSV file represents a given latitude, and each column represents a given longitude.

In the CSV file, the land cell must either have negative or `NA` values.

! Warning

The land cells (`NA`values), the number of lines (longitudes) and columns (latitudes) must be consistent with the lower trophic level (LTL) forcing file!

6.2.2 Accessibility and catchability matrix

Accessibility and catchability matrix are provided as a CSV file. The first line specifies the name of the predator (either fish species or fishing gear), while the first column specifies the name of the prey (either fish species or biogeochemical forcing).

In accessibility matrix, the size information is provided by appending the `<` character before the upper bound of the class value.

For example, if the predator column is `cod` , the values will be used for all `cod` schools. On the other hand, if the CSV file contains three columns:

cod < 0.25	cod < 2	cod
0.25	0.5	0.7

The `0.25` value will be used for `cod` schools smaller than `0.25` cm, the `0.5` value will be used for `cod` schools between 0.25 and 2cm, and the `0.7` value will be used for all the other school size.

In catchability matrix (used for fishing mortality and discards when fisheries are enabled), the predator names are the fishing gears.

6.2.3 Time series

Time series are provided in CSV files that contain two columns and a header. The first column contains the time value, which are not used by Osmose. The second column contains the values that will be used. Below is an example of CSV time series:

Table 6.2: Example of time series CSV file

Time	Value
0	0.05
0.083333336	0.15
0.16666667	0.15
0.25	0.15
0.33333334	0.05
0.41666666	0
0.5	0
0.5833333	0
0.6666667	0
0.75	0
0.8333333	0
0.9166667	0

If the number of values is less than the total number of simulation time steps, the same values will be repeated over and over. For example, in a `100` year simulation with a bi-monthly time-step (`24` time steps per year), the user can either provide `2400` values (one value per time step) or `24` values (the same values will be used `100` times).

6.2.4 By class time series

6.3 Forcing data

Secondly, the modelled system is driven by prey fields which are not explicitly represented as focus species in OSMOSE. For example, biomass fields of phytoplankton and zooplankton varying in space and time are typical input to OSMOSE. In the different OSMOSE applications, these prey fields were usually produced from coupled hydrodynamic and biogeochemical (BGC) models such as ROMS-NPZD (Travers-Trolet, Shin, and Field 2014), ROMS-PISCES (Oliveros Ramos 2014), NEMOMed-ECO3M (Halouani et al. 2016) or are derived from observational data (Grüss et al. 2015; Fu et al. 2013). Benthic resources can also drive the dynamics of the system as in Halouani et al. (2016).

References

Fu, Caihong, R Ian Perry, Yunne-Jai Shin, Jake Schweigert, and Huizhu Liu. 2013. "An Ecosystem Modelling Framework for Incorporating Climate Regime Shifts into Fisheries Management." *Progress in Oceanography* 115: 53–64. https://doi.org/10.1016/j.pocean.2013.03.003.

Grüss, Arnaud, Michael J Schirripa, David Chagaris, Michael Drexler, James Simons, Philippe Verley, Yunne-Jai Shin, Mandy Karnauskas, Ricardo Oliveros-Ramos, and Cameron H Ainsworth. 2015. "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. https://doi.org/10.1016/j.jmarsys.2014.11.004.

Halouani, Ghassen, Frida Ben Rais Lasram, Yunne-Jai Shin, Laure Velez, Philippe Verley, Tarek Hattab, Ricardo Oliveros-Ramos, et al. 2016. "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. https://doi.org/10.1016/j.ecolmodel.2016.08.008.

Oliveros Ramos, Ricardo. 2014. "End-to-End Modelling for an Ecosystem Approach to Fisheries in the Northern Humboldt Current Ecosystem." PhD thesis, University of Montpellier, France.

Travers-Trolet, M, Y-J Shin, and JG Field. 2014. "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. https://doi.org/10.2989/1814232X.2014.883326.