Multi-species size-based ecological modelling in R (`mizer`): An introductory vignette

F. Scott[*], J. Blanchard[**] and K. H. Andersen[***]

[*]Cefas, Lowestoft, UK
[**]University of Sheffield, UK
[***]DTU Aqua, Copenhagen, DK

`mizer` version 0.0 , 2012-12-14

Contents

1 Summary **3**

2 Introduction **3**
2.1 Installation . . . 3
2.2 Methods and classes in R . . . 3

3 The principles of size-based modelling **3**
3.1 The model . . . 4

4 Using `mizer` **5**

5 Setting the model parameters using *MizerParams* **5**
5.1 The species parameters . . . 6
5.2 Fishing gears and selectivity . . . 6
5.3 The stock-recruitment relationship . . . 7
5.4 The interaction matrix . . . 8
5.5 The other `MizerParams()` arguments . . . 8
5.6 Putting all this together: an example of making a simple *MizerParams* object . . . 8
5.7 Setting different gears . . . 12

6 Running a simulation **13**
6.1 The time arguments . . . 13
6.2 Setting the fishing effort . . . 13
6.3 Setting the initial population abundance . . . 14
6.4 What do you get from running `project()`? . . . 14
6.5 Projection examples . . . 14
6.5.1 Projections with simple effort arguments . . . 15
6.5.2 An example of changing effort through time . . . 16

7 Exploring the simulation results **17**
7.1 Summary methods for *MizerSim* objects . . . 17
7.1.1 Examples of using the summary methods . . . 18
7.2 Methods for calculating indicators . . . 19
7.2.1 Examples of calculating indicators . . . 19
7.3 Plotting the results . . . 20
7.3.1 Plotting examples . . . 20

8 A more detailed example using the North Sea **20**

9 Under the hood of `mizer`: notes for developers **23**

10 Acknowledgements **23**

[1]This document is included as a vignette (a LaTeX document created using the R function `Sweave`) of the package `mizer`. It is automatically dowloaded together with the package and can be accessed through R typing `vignette("mizer_vignette")`.

1 Summary

`mizer` is a package for R that implements size-based ecological models. The package has been developed to model marine ecosystems that are subject to fishing. However, it may also be appropriate for other ecosystems. The package contains routines and methods to allow users to set up the model community, and then project it through time under different fishing strategies. Methods are included to explore the results, including plots and calculation of community indicators such as the slope of the size spectrum. Size-based models can be complicated so `mizer` contains many default options that can be easily changed by the user.

2 Introduction

2.1 Installation

R is a open source software project and can be freely downloaded from the CRAN website. There are already several resources for an introduction to R . The CRAN website link to contributed documentation also offers excellent introductions in several languages. Once R is running the installation of additional packages is quite straightforward. To install the `mizer` package from R simply type:

```
> install.packages("mizer")
```

Once the `mizer` package is installed, it needs to be made accessible to the current R session by the command:

```
> library(mizer)
```

For an overview of the package documentation or the details of a particular command (such as the function `project`) you can type:

```
> help(package="mizer")
> help(project)
```

The first command gives a brief summary of the available commands in the package, the second give more detailed information about a specific command.

2.2 Methods and classes in R

This is only a brief reminder that expressions in R manipulate objects, which may be data (a scalar or a matrix for example), but objects may also be functions, or more complex collections of objects. All objects have a class, which enables functions to act on them appropriately. Thus, for example, when the function `summary` is applied on an object of class *MizerParams* (the class that stores the model parameters), it would act differently than on a simple matrix.

3 The principles of size-based modelling

Describe the basic principles of size-based modelling. Abundance of individuals decreases with body size leading to a 'size spectrum'. Big things eat little things.

References to: Sheldon and Parsons (1967); Andersen and Beyer (2006); Andersen and Pedersen (2009); Blanchard; NORTH SEA PAPER etc.

Food dependent growth.

Changes in the size spectrum as an indicator (LFI, MW, slope).

3.1 The model

Describe the model here. Assumptions

Check equations and notations are correct!

Prey size selection:

$$\phi\left(\frac{w_{prey}}{w}\right) = exp\left[-\left(ln\left(\frac{\beta w_{prey}}{w}\right)\right)^2/(2\sigma^2)\right] \tag{3.1}$$

Search rate:

$$V(w) = \gamma w^n \tag{3.2}$$

Encountered food:

$$E(w) = V(w)\sum_j \theta_{ij}\int_0^\infty N_j(w_p)w_p dw_p \tag{3.3}$$

Maximum consumption:

$$I_{max} = h_i w^n \tag{3.4}$$

Feeding level:

$$f_i(w) = \frac{E_i(w)}{E_i(w) + I_{max}} \tag{3.5}$$

Maturation:

$$\psi(w) = \left[1 + \left(\frac{w}{w_i^*}\right)^{-10}\right]^{-1}\left(\frac{w}{W_i}\right)^{1-n} \tag{3.6}$$

Somatic growth:

$$g_i(w) = (\alpha f_i(w) h_i w^n - k_i w^p)(1 - \psi(w)) \tag{3.7}$$

Gonadal growth:

$$g_r(w) = (\alpha f_i(w) h_i w^n - k_i w^p)\psi(w) \tag{3.8}$$

Density independent recruitment:

$$\varepsilon/(2w_0 N_i(w_0) g(w_0))\int_{w_i^*}^{w_i} N_i(w) g_r(w) dw \tag{3.9}$$

Background mortality:

$$\mu_0 = Z_0 W^Z \tag{3.10}$$

Predation mortality:

$$\mu_{p,i}(w) = \sum_j \int_{w_0}^\infty \phi\left(\frac{w'}{w}\right)(1 - f_j(w'))V(w')\theta_{i,j}N_j(w')dw' \tag{3.11}$$

Growth of the resource spectrum:

$$\frac{\delta N_r(w)}{\delta t} = r_0 w^{n-1}(k(w) - N_r(w)) - \mu_{p,r}(w)N_r(w) \tag{3.12}$$

Carrying capacity of the resource spectrum:

$$k(w) = k_r^{-\lambda} \tag{3.13}$$

Interaction matrix

etc

Just checking Bibtex works in vignette (Andersen and Pedersen, 2010).

4 Using `mizer`

Using `mizer` is relatively simple. There are two main stages:

- Setting the model parameters. This is done by creating an object of class *MizerParams*.
- Running a simulation. This is done by calling the `project()` method on the model parameters. This produces an object of *MizerSim* which contains the results of the simulation.

After a simulation has been run, the results can be explored using a range of plots and summaries. These stages are explained in the following sections.

5 Setting the model parameters using *MizerParams*

Before any simulations can be run it is necessary to set the parameters of the model. This is probably the most complicated part of the using the `mizer` package so we will take it slowly.

The `mizer` package has its own class for storing model parameters: *MizerParams*. This class stores the:

- life-history parameters of the species in the community, such as W_∞;
- size-based biological parameters for the species, such as the search volume;
- stock-recruitment relationship functions and parameters of the species;
- interaction matrix to describe the spatial overlap of pairs of species;
- parameters relating to the growth and dynamics of the background resource spectrum;
- fishing gear parameters: selectivty and catchability.

Note that the *MizerParams* class does not store any parameters that can vary through time, such as fishing effort or population abundance. These are stored in the *MizerSim* class which we will come to later in Section 6.

Although the *MizerParams* class seems complicated, it is relatively straightforward to set up and use. Objects of class *MizerParams* are created using the constructor method `MizerParams()`. This

constructor method can take many arguments. However, creation is simplified because many of the arguments have default values.

In the rest of this section we look at the main arguments to the `MizerParams()` constructor method. To help understand how the constructor is used and how the *MizerParams* class relates to the equations given in Section 3, there is an example section where we create example parameter objects using data that comes with the `mizer` package.

5.1 The species parameters

Although many of the arguments used when creaing a *MizerParams* object are optional, there is one argument that must be supplied by the user: the species specific parameters. For each species in the model community you must supply the life-history parameters, the selectivity function parameters and the stock-recruitment function parameters.

The species parameters must all be stored in a single data.frame object. Each column of the parameter data.frame is a parameter and each row has the parameters for one of the species in the model. Although it is possible to create the data.frame by hand in R , it is probably easier to create the data as a .csv file (perhaps using a suitable open source spreadsheet such as LibreOffice) and then read the data into R .

There are some essential columns that you must include in the parameter data.frame. Some columns have default values, so that if you do not include them in the species parameter data.frame, they will be automatically added when the *MizerParams* object is created. A description of the columns of the species parameter data.frame and any default values can be seen Table 1.

You can see in Table 1 that most of the species specific parameters relate to the life history of the species. The others relate to the gear selectivity function and the stock-recruitment relationship. These are explained further in Sections 5.2 and 5.3 respectively.

5.2 Fishing gears and selectivity

In this section we take a look at how fishing is implemented and how fishing gears are set up within `mizer`.

In `mizer`, fishing mortality is imposed on species by fishing gears. The fishing mortality F imposed by gear g on species s at size w is calculated as:

$$F_{s,g,w} = S_{s,g,w} Q_{s,g} E_g \tag{5.1}$$

where S is the selectivity by species, gear and size, Q is the catchability by species and gear and E is the fishing effort by gear. The selectivity at size has a range between 0 (not selected at that size) to 1 (fully selected at that size). Catchability is used as an additional scalar to make the link between gear selectivity, fishing effort and fishing mortality. For example, it can be set so that an effort of 1 gives a desired fishing mortality. In this way effort can then be specified relative to a 'base effort', e.g. the effort in a particular year. Fishing effort is set when the simulation is run and can vary through time (see Section 6).

At the moment a species can only be selected by one fishing gear, although each gear can select more than one species (this is a limitation with the current package that will be developed in future releases).

The selectivity at size of each gear is given by a selectivity function. Some selectivity functions are included in the package. New functions can be defined by the user. Each gear has the same selectivity function for all the species it selects, but the parameter values for each species may be different.

The name of the selectivity function is given by the *sel_func* column in the species parameters data.frame. Each selectivity function has a range of arguments. Values for these arguments must be included as columns in the species parameters data.frame. The names of the columns must exactly match the names of the arguments. For example, the default selectivity function is `sigmoid_length`. The arguments for this selectivity function can be seen in the help page for this function. To see them enter:

```
> ?sigmoid_length
```

It can be seen that the `sigmoid_length` function has arguments `w`, `l25`, `l50`, `a` and `b`. The first argument, `w`, is size (the function calculates selectivity at size). All selectivity functions must have `w` as the first argument. The values for the other arguments must be found in the species parameters data.frame. This can be seen in the example in Section 5.6. If they are not there, an error is thrown when the *MizerParams* object is created.

Users are able to write their own size based selectivity function. The first argument to the function must be `w` and the function must return a vector of the selectivity (between 0 and 1) at size.

The name of the fishing gear is given in the *gear* column of the species parameter data.frame. If the *gear* column is not specified, the default gear name is simply the name of the species, i.e. each species is fished by a different gear.

5.3 The stock-recruitment relationship

The stock-recruitment relationship in `mizer` means something slightly different to the standard fisheries terminology. In standard fisheries science, recruitment means the abundance of new recruits entering the fishery (e.g. the number of age 1 added to the population each year). This recruitment is calculated by the stock-recruitment relationship, based on some measure of the reproductive potential of the stock, such as spawning stock biomass.

In `mizer` recruitment means the number of individuals that enter the size-spectrum at the smallest size group of that species (given by the parameter *w_min* in the species parameter data.frame). As can be seen in Section 3, calculating the recruitment involves calculating the 'density independent' recruitment, RDI (equation 3.9). The RDI is then modified by a stock-recruitment relationship to impose some form of density-dependence. This then results in the density-dependent recruitment, RDD. Without this density dependence, the realised recruitment to the smallest size class is determined only by RDI.

The default stock-recruitment relationship is a Beverton-Holt type function which has the form:

$$RDD = \frac{R_{max}.RDI}{R_{max} + RDI} \tag{5.2}$$

where R_{max} is a parameter which determines the maximum recruitment.

Similar to the fishing selectivity functions, any parameter used in the stock-recruitment function, other than RDI, must be in the species parameter data.frame and the column must have the same name as the function argument. For example, if the above default stock-recruitment function is to be used then the species parameter data.frame must have a R_{max} column.

Users are able to write their own stock-recruitment function. The first argument to the function must be `rdi`.

5.4 The interaction matrix

The interaction matrix describes the interaction of each pair of species in the model. This can be viewed as a proxy for spatial interaction e.g. to model predator-prey interaction that is not size based. The values in the interaction matrix are used to scale the size-based predation pressure. The matrix is square with every element being the interaction between a pair of species. The dimensions, nrows and ncolumns, therefore equal the number of species. The values are between 0 (species do not overlap and do not feed on each other) to 1 (species overlap perfectly). If all the values in the interaction matrix are set to 1 then predator-prey interactions are determined entirely by size-preference.

The interaction matrix must be of type `array` or `matrix`. One way of creating your own is to enter the data using a spreadsheet (such as LibreOffice) and saving it as a .csv file. The data can be read into R using the commmand `read.csv()`. This reads in the data as a data.frame. We then need to convert this to a matrix using the `as()` function. An example of how to do this is given in Section 5.6.

It should be noted that the order of species in the interaction matrix has to be the same as the order in the species parameters data.frame. Although you can specify the dimnames of the interaction matrix, these names are overwritten by the species names of the species parameters data.frame inside the *MizerParams* constructor.

If an interaction matrix is not specified to the `MizerParams()` contructor the default interation matrix is used. This all values set to 1.

5.5 The other `MizerParams()` arguments

As well as the essential species parameters data.frame and the interaction matrix, there are several other arguments to the `MizerParams` constructor. These have default values. Although it is tempting to just charge on and use the default values, many of the values come from a particular model for the North Sea and so may not be appropriate for what you are doing. We'll look at these arguments here, describe what they do and discuss how to choose appropriate ones for your model (TODO). The arguments can be seen in Table 2.

How to choose parameter values.

5.6 Putting all this together: an example of making a simple *MizerParams* object

As mentioned in the preceeding sections, an object of *MizerParams* is created by using the `MizerParams()` constructor method. This takes the following arguments:

object The species parameter data.frame (see Section 5.1). This is compulsory with no default value.

inter The interaction matrix (see Section 5.4).

... Other model parameters (see Section 5.5).

In the rest of this section we demonstrate how to pull these elements together to make *MizerParams* objects.

The first step is to prepare the species specific parameter data.frame. As mentioned above, one way of doing this is to use a spreadsheet and save it as a .csv file. We will use this approach here. An example .csv file has been included in the package. This file is placed in the *doc* folder of the package installation. The location of the file can be found by running:

```
> system.file("doc/simple_species_params.csv",package="mizer")
```

This file can be opened with most spreadsheets for you to inspect. This can be loaded into R using the following code (after you have told R to look in the right directory):

```
> params_data <- read.csv("simple_species_params.csv")
```

This reads the .csv file into R in the form of a data.frame. You can check this with the `class`:

```
> class(params_data)
```

```
[1] "data.frame"
```

The example data.frame can be inspected by entering the name of the object.

```
> #head(params_data)
> params_data
```

```
   species   w_inf       w_mat     beta sigma         h       ks    gamma
1    Sprat    63.9    2.673344  51076.0   0.8  8.095779 1.619156 1.43e-11
2  Sandeel    23.4   16.116931 398849.0   1.9 12.728150 2.545630 7.18e-12
3   N.pout   232.2    7.383764     21.5   1.5 10.346438 2.069288 2.74e-11
4  Herring   296.7   42.926512 280540.0   3.2 18.953044 3.790609 6.65e-12
5      Dab   254.2   21.464400    191.0   1.9 26.552898 5.310580 4.15e-11
6  Whiting   888.8   59.180806     22.0   1.5 19.950454 3.990091 5.27e-11
7     Sole   566.0  141.404189    381.0   1.9 20.679760 4.135952 2.95e-11
8  Gurnard   570.7   91.688005    283.0   1.8 11.543509 2.308702 1.81e-11
9   Plaice  2525.0  274.450006    113.0   1.6 11.574602 2.314920 2.30e-11
10 Haddock  4316.5  226.882937    558.0   2.1 21.166738 4.233348 2.60e-11
11     Cod 39851.3 1243.302011     66.0   1.3 42.981002 8.596200 1.13e-10
12  Saithe 39658.6 2538.845804     40.0   1.1 22.166263 4.433253 7.37e-11
           z0    l25    l50        a     b        r_max
1  0.15007821  7.008  7.291 0.007000 3.018 5.430000e+11
2  0.20977125 13.967 15.841 0.001243 3.320 3.650000e+11
3  0.09761814  8.474  9.616 0.009000 2.962 1.510000e+12
4  0.08995916  0.857 11.529 0.002000 3.402 4.400000e+12
5  0.09471659 16.068 26.370 0.009000 3.032 3.340000e+11
6  0.06240459 21.970 27.855 0.005000 3.143 2.673956e+10
7  0.07253469 22.944 27.268 0.006000 3.098 1.158950e+09
8  0.07233502 21.970 27.855 0.004000 3.202 1.267022e+10
9  0.04406199 16.068 26.370 0.007000 3.108 1.000000e+18
10 0.03685027 21.209 24.642 0.005000 3.171 6.196713e+10
11 0.01756590 15.064 20.685 0.005000 3.156 5.263727e+08
12 0.01759431 46.504 52.006 0.006000 3.093 1.651426e+10
```

This data set is for a model of the North Sea (REFERENCE). There are 12 species. As you can see, we have all the essential life history columns as described in Table 1.

There is no $sel_f unc$ column to determine the selectivity function. This means that the default selectivity function, `sigmoid_length`, will be used. However, as mentioned in Section 5.2, this function

also needs several other arguments. The species data.frame therefore already includes the columns a, b, $l25$ and $l50$ for the default selectivity function.

Similarly, the column *r_max* is the parameter for the default stock-recruitment function described in Section 5.3.

The values of the non-essential species specific parameters *alpha* and *erepro*, are not included in the data.frame. This means that the default values will be automatically used when we create the *MizerParams* object.

Strictly speaking, this species parameter data.frame is the minimum we need to create a *MizerParams* object. However, the model would be improved by also including a case study specific interaction matrix.

An example interaction matrix has been included in `mizer` as a .csv file. The location of the file can be found by running:

```
> system.file("doc/inter.csv",package="mizer")
```

Take a look at it in a spreadsheet if you want. As mentioned above, to read this file into R we can make use of the `read.csv()` function. However, this time we want the first column of the .csv file to be the row names. We therefore use an additional argument to the `read.csv()` function: `row.names`.

```
> inter <- read.csv("inter.csv", row.names=1)
```

The `read.csv()` function reads the data into a data.frame. We want the interaction matrix to be of class *matrix* so we need to make use of the `as()` function.

```
> inter <- as(inter, "matrix")
```

We now have the species parameters data.frame and the interaction matrix. To make the *MizerClass* object you just call the constructor method and pass in the arguments. We will use default values for the remainder of the arguments to the `MizerParams()` method. These are given in Table 2. This means that we only need to pass in two arguments to the constructor:

```
> params <- MizerParams(params_data, interaction = inter)
```

Note that the first argument must be the species parameters data.frame. The remaining arguments can be in any order but should be named. If we didn't want to use default values for the other arguments we would pass them in to the constructor by name. For example, if wanted our species size spectrum to have 200 size bins instead of 100 (the default) we would use:

```
> params <- MizerParams(params_data, interaction = inter, no_w = 200)
```

The above command creates an object of class *MizerParams*. But what does that mean? The method uses the parameters specified in the arguments and the default values to implement some of the equations described in Section 3 and calculate parameters for the model.

As mentioned above an object of class *MizerParams* stores a wide range of model parameters. This information is held in what are known as 'slots'. A description of these slots can be found by calling `help()` on the class:

```
> help("MizerParams-class")
```

A list of the slots can also be seen using the `slotNames()` method. The different slots can be accessed using the `@` operator. For example, to look at the search volume slot, `search_vol` you enter (the results are not shown here as the array is pretty huge):

```
> params@search_vol
```

The slot *species_params* contains the species parameters data.frame that was passed in to the constructor. Now it also contains any default values (for example, *alpha*) that may not have been included in the original data.frame.

Also note how the default fishing gears have been set up. Because we did not specify gear names in the original species parameter data.frame, each species is fished by a unique gear named after the species. This can be seen in the new *gear* column which holds the names of the fishing gears. Also, the selectivity function for each fishing gear has been set in the sel_func column (function `sigmoid_length()`).

WHY IS CATCHABILITY ALSO IN THERE? REMOVE

PLOT SELECTIVITY - ADD METHOD?

```
> params@species_params
```

```
   species   w_inf       w_mat     beta sigma         h       ks    gamma
1    Sprat    63.9    2.673344  51076.0   0.8  8.095779 1.619156 1.43e-11
2  Sandeel    23.4   16.116931 398849.0   1.9 12.728150 2.545630 7.18e-12
3   N.pout   232.2    7.383764     21.5   1.5 10.346438 2.069288 2.74e-11
4  Herring   296.7   42.926512 280540.0   3.2 18.953044 3.790609 6.65e-12
5      Dab   254.2   21.464400    191.0   1.9 26.552898 5.310580 4.15e-11
6  Whiting   888.8   59.180806     22.0   1.5 19.950454 3.990091 5.27e-11
7     Sole   566.0  141.404189    381.0   1.9 20.679760 4.135952 2.95e-11
8  Gurnard   570.7   91.688005    283.0   1.8 11.543509 2.308702 1.81e-11
9   Plaice  2525.0  274.450006    113.0   1.6 11.574602 2.314920 2.30e-11
10 Haddock  4316.5  226.882937    558.0   2.1 21.166738 4.233348 2.60e-11
11     Cod 39851.3 1243.302011     66.0   1.3 42.981002 8.596200 1.13e-10
12  Saithe 39658.6 2538.845804     40.0   1.1 22.166263 4.433253 7.37e-11
           z0    l25    l50        a     b        r_max    gear k alpha erepro
1  0.15007821  7.008  7.291 0.007000 3.018 5.430000e+11   Sprat 0   0.6      1
2  0.20977125 13.967 15.841 0.001243 3.320 3.650000e+11 Sandeel 0   0.6      1
3  0.09761814  8.474  9.616 0.009000 2.962 1.510000e+12  N.pout 0   0.6      1
4  0.08995916  0.857 11.529 0.002000 3.402 4.400000e+12 Herring 0   0.6      1
5  0.09471659 16.068 26.370 0.009000 3.032 3.340000e+11     Dab 0   0.6      1
6  0.06240459 21.970 27.855 0.005000 3.143 2.673956e+10 Whiting 0   0.6      1
7  0.07253469 22.944 27.268 0.006000 3.098 1.158950e+09    Sole 0   0.6      1
8  0.07233502 21.970 27.855 0.004000 3.202 1.267022e+10 Gurnard 0   0.6      1
9  0.04406199 16.068 26.370 0.007000 3.108 1.000000e+18  Plaice 0   0.6      1
10 0.03685027 21.209 24.642 0.005000 3.171 6.196713e+10 Haddock 0   0.6      1
11 0.01756590 15.064 20.685 0.005000 3.156 5.263727e+08     Cod 0   0.6      1
12 0.01759431 46.504 52.006 0.006000 3.093 1.651426e+10  Saithe 0   0.6      1
         sel_func w_min w_min_idx
1  sigmoid_length 0.001         1
2  sigmoid_length 0.001         1
3  sigmoid_length 0.001         1
4  sigmoid_length 0.001         1
```

```
5  sigmoid_length 0.001         1
6  sigmoid_length 0.001         1
7  sigmoid_length 0.001         1
8  sigmoid_length 0.001         1
9  sigmoid_length 0.001         1
10 sigmoid_length 0.001         1
11 sigmoid_length 0.001         1
12 sigmoid_length 0.001         1
```

There is a `summary()` method for *MizerParams* objects which prints a useful summary of the model parameters:

```
> summary(params)
```

5.7 Setting different gears

In the above example, each species is caught by a different gear (named after the species it catches). This is the default when there is no *gear* column in the species parameter data.frame.

Here, we look at an example where we do specify the fishing gears. We take the original *params_data* species parameter data.frame that was read in above and bind an additional column, *gear*, to it. This *gear* column contains the name of the gear that catches the species in that row. Here we set up four different gears: Industrial, Pelagic, Beam Trawl and Otter Trawl, that catch different combinations of species.

```
> #params_data <- read.csv("../inst/doc/simple_species_params.csv")
> params_data$gear <- c("Industrial","Industrial","Industrial",
+                       "Pelagic","Beam Trawl","Otter Trawl",
+                       "Beam Trawl","Otter Trawl","Beam Trawl",
+                       "Otter Trawl","Otter Trawl","Otter Trawl")
```

We then make a new *MizerParams* object as before:

```
> params_gears <- MizerParams(params_data, interaction = inter)
```

You can see that the gears in the mew *MizerParams* object have been set correctly by examining the *catchability* slot:

```
> params_gears@catchability
```

```
             sp
gear          Sprat Sandeel N.pout Herring Dab Whiting Sole Gurnard Plaice
  Industrial      1       1      1       0   0       0    0       0      0
  Pelagic         0       0      0       1   0       0    0       0      0
  Beam Trawl      0       0      0       0   1       0    1       0      1
  Otter Trawl     0       0      0       0   0       1    0       1      0
             sp
gear          Haddock Cod Saithe
  Industrial        0   0      0
  Pelagic           0   0      0
  Beam Trawl        0   0      0
  Otter Trawl       1   1      1
```

A catchability of 0 means that a species is not caught by that gear.

In this example the same gear now catches multiple stocks. For example, the 'Industrial' gear catches Sprat, Sandeel and Norway Pout. Why would we want to set up the gears like this? (Note that we are still using the default *sigmoid_length* selectivity function for all the gears in this example.)

In the next section we will see that to project the model through time you need to specify the fishing effort for each gear through time. By setting the gears up in this way you can run different management scenarios of changing the efforts of the fishing gears rather than on individual species. It also means that after a simulation has been run you can examine the catches by gear.

6 Running a simulation

In `mizer` simulations are performed using the `project()` method. This method takes a *MizerParams* object and projects it forward through time, starting from an initital population abundance and with a pre-determined fishing effort pattern.

Running a projection with `project()` requires various arguments:

A *MizerParams* object The model parameters (see previous section);

Fishing effort The fishing effort of each gear through time;

Initial population The initial abundances of the stocks and the background spectrum;

Time arguments Arguments to control the time of the simulation, including the simulation time step, the length of the simulation and how frequently the output is stored.

The help page for `project()` describes the arguments in more detail.

The *MizerParams* class was explored in the previous section. In this section we will look at the other arguments and use examples to perform some simple projections.

6.1 The time arguments

There are three arguments that control the `project()` method: *dt*, *t_max* and *t_save*.

dt is the time step used by the numerical solver in `project()`. This is the time step on which the model operates. SOMETHING ABOUT THE IMPACT THAT DT HAS ON THE SIMULATION. *t_max* determines the maximum time of the simulation, i.e. how long the projection is run for. Note that this is not necessarily the same as the number of time steps in the model. The number time steps in the model is given by *t_max* / *dt*. For example, a simulation with $dt = 0.5$ and $t\_max = 10$ will perform 20 time steps. The default values for *dt* and *t_max* are 1 and 100 respectively. The final argument is *t_save*. This sets how frequently `project()` stores the state of the model in the *MizerSim* object. For example, if $t\_save = 2$, the state of the model is stored at t = 0, 2, 4... etc. *t_save* must be a multiple of *dt*. The default value of *t_save* is 1.

6.2 Setting the fishing effort

The fishing effort argument describes the effort of the fishing gears in the model through time. Information on the fishing gears and their selectivities and catchabilities is stored in the `MizerParams` argument.

There are three ways of setting the fishing effort. Examples of all three can be seen in Section 6.5.

The simplest way is by passing the $effort$ argument as a single number. This value is then used as the fishing by all of the gears at each time step of the projection, i.e. fishing effort is constant throughout the simulation and is the same for all gears. The length of the simulation is determined by the *t_max* argument (see Section 6.1).

Another way of setting the fishing effort is to use a numeric vector that has the same length as the number of gears. The values in the vector are used as the fishing effort of each gear at each time step, i.e. again, the fishing effort is constant through time but each gear can have a different constant effort. The order of values in the effort vector must be the same as the order of gears in the `MizerParams` object (it may be helpful to name the effort vector but these names are not used in `project()`). Again, the length of the simulation is determined by the *t_max* argument.

Finally, the most sophisticated way of setting the fishing effort is to use a two-dimensional array of values, set up as time step by gear. Each row of the array has the effort values of each fishing gear at a time step (the time step is set by the argument *dt*, see above). This means that it is not necessary to supply a *t_max* argument in this case because the maximum time of the simulation is calculated by the number of the time steps in the effort array (the first dimension) and the value of *dt*. If a value for *t_max* is also supplied it is ignored.

6.3 Setting the initial population abundance

When running a simulation with the `project()` method, the initial populations of the species and the background spectrum need to be specified. These are passed to `project()` as the arguments *initial_n* and *initial_n_pp* respectively. *initial_n* is a matrix (with dimensions species x size) that contains the initial abundances of each species at size (the sizes match those in the species size spectrum). *initial_n_pp* is a vector of the same length as the the length of the full spectrum.

By default, the *initial_n* argument has values calculated using the methods described in NORTH SEA PAPER. These are implemented in the function `getInitialN()`. The default value for *initial_n_pp* is the carrying capacity of the background spectrum, stored in the *cc_pp* slot of the *MizerParams* parameters object.

6.4 What do you get from running `project()`?

Running `project()` returns an object of type *MizerSim*. Like the *MizerParams* class this has various slots to contain the output of the simulations. An object of *MizerSim* has four slots, details of which can be seen in the help page. The *params* slot holds the *MizerParams* object that was passed in to `project()`. The $effort$ slot holds the fishing effort of each gear through time. Note that the $effort$ slot may not be exactly the same as the one passed in as the $effort$ argument. This is because only the saved effort is stored (the frequency of saving is determined by the argument *t_save*). The *n* and *n_pp* hold the saved abundances of the species and the background population at size respectively. Note that The *n* and *n_pp* slots have one more row than the $effort$ slot. This is to the store the initial populations.

6.5 Projection examples

In this section we'll look at how to run simulations with the `project()` method whilst specifying fishing effort in different ways. We will use the *params* object with four gears that was created in the *MizerParams* example above.

6.5.1 Projections with simple effort arguments

In this first example we will use constant effort through time for each of the fishing gears and this effort will be used for all the gears. This means we only need to specify the $effort$ argument as a single numeric. As well as thinking about the $effort$ argument we also need to consider the time parameters. We will project the populations forward until time equals 10 ($t\_max = 10$), with a time step of 1 ($dt = 1$), saving the output every time step ($t\_save = 1$).

```
> sim <- project(params_gears, effort = 1, t_max = 10, dt = 1, t_save = 1)
```

The resulting *sim* object is of class *MizerSim*. The species abundances at size through time can be seen in the n slot. This is an array (time x species x size). Consequently, this array can get very big so inspecting it can be difficult. A range of summary methods and plots are available to make this job easier. We will look at those in Section 7. In the example we have just run the time dimension of the n slot has 11 rows (one for the initial population and then one for each of 10 saved time steps). There are also 12 species each with 100 sizes. We can check this by running the `dim()` function and looking at the dimensions of the n array:

```
> dim(sim@n)
```

```
[1]  11  12 100
```

To pull out the abundances of a particular species through time at size you can subset the array. For example to look at Cod through time you can use:

```
> sim@n[,"Cod",]
```

The effort through time can be inspected by looking at the $effort$ slot (we use the `head()` function to just show the first few lines). In this example, we specified the $effort$ argument as a single numeric of value 1. As you can see this results in the same effort being used for all gears for all time steps:

```
> head(sim@effort)
```

```
    gear
time Industrial Pelagic Beam Trawl Otter Trawl
   1          1       1          1           1
   2          1       1          1           1
   3          1       1          1           1
   4          1       1          1           1
   5          1       1          1           1
   6          1       1          1           1
```

If we decrease dt and t_save but keep t_max the same then we can see that the time dimension of the $effort$ slot changes accordingly. Now the model time step is 0.5 and we store the output every at multiples of time = 0.5. This will also be true of the n and n_pp slots.

```
> sim <- project(params_gears, effort = 1, t_max = 10, dt = 0.5, t_save = 0.5)
> head(sim@effort)
```

```
     gear
time  Industrial Pelagic Beam Trawl Otter Trawl
  0.5          1       1          1           1
  1            1       1          1           1
  1.5          1       1          1           1
  2            1       1          1           1
  2.5          1       1          1           1
  3            1       1          1           1
```

As mentioned above, we can also set the effort values for each gear seperatley using a vector of effort values. This still keeps the efforts constant through time but it means that each gear can have a different constant effort.

For example, if we want to switch off the industrial gear (i.e. effort = 0), keep the pelagic gear effort at 1, and set the effort of both the trawl gears at 0.5 we set the effort for each gear using a single effort vector, where the positions of elements in the vector match the position of the gears:

```
> effort <- c(0,1,0.5,0.5)
```

We then call `project()` with this effort and inspect the resulting $effort$ slot (again we use the `head()` function to just show the first few lines):

```
> sim <- project(params_gears, effort = effort, t_max = 10, dt = 1, t_save = 1)
> head(sim@effort)
```

```
    gear
time Industrial Pelagic Beam Trawl Otter Trawl
   1          0       1        0.5         0.5
   2          0       1        0.5         0.5
   3          0       1        0.5         0.5
   4          0       1        0.5         0.5
   5          0       1        0.5         0.5
   6          0       1        0.5         0.5
```

6.5.2 An example of changing effort through time

In this example we set up a more complicated fishing effort structure that allows the fishing effort of each gear to change through time. As mentioned above, to do this effort must be supplied as a two dimensional array or matrix. The first dimension is time step (determined by the dt argument and the second dimension is gear. Also, as mentioned above, if effort is passed in as an array or matrix then the length of the simulation is determined by the number of rows in the effort array and dt; the argument t_max is not used. In our example $params_gears$ object we have 4 gears.

In this example, the effort of the industrial gear is held constant at 0.5, the effort of the pelagic gear is increased linearly from 1 to 2, the effort of the beam trawl decreases linearly from 1 to 0, whilst the effort of the otter trawl decreases linearly from 1 to 0.5. The final dimension of the array is 10 by 4.

```
> effort_array <- cbind(rep(0.5,10),
+     seq(from = 1, to = 2, length = 10),
+     seq(from = 1, to = 0, length = 10),
+     seq(from = 1, to = 0.5, length = 10))
```

The first few rows of the effort array are shown as an illustration:

```
> head(effort_array)
```

```
     [,1]     [,2]      [,3]      [,4]
[1,]  0.5 1.000000 1.0000000 1.0000000
[2,]  0.5 1.111111 0.8888889 0.9444444
[3,]  0.5 1.222222 0.7777778 0.8888889
[4,]  0.5 1.333333 0.6666667 0.8333333
[5,]  0.5 1.444444 0.5555556 0.7777778
[6,]  0.5 1.555556 0.4444444 0.7222222
```

The effort array does not need to have dimension names because it is the order of the columns that is important, not their name. The order of the columns must match the order of the gears in the *MizerParams* object. As you can see in the previous examples, the dimensions of the effort array do get named when it is processed by `project()`.

When running `project()` with an effort array we do not specify the *t_max* argument. The duration of the projection is given by *dt* and the number of rows in the effort array.

```
> sim <- project(params_gears,effort=effort_array, dt=1, t_save = 1)
> head(sim@effort)
```

```
    gear
time Industrial  Pelagic Beam Trawl Otter Trawl
   1        0.5 1.000000  1.0000000   1.0000000
   2        0.5 1.111111  0.8888889   0.9444444
   3        0.5 1.222222  0.7777778   0.8888889
   4        0.5 1.333333  0.6666667   0.8333333
   5        0.5 1.444444  0.5555556   0.7777778
   6        0.5 1.555556  0.4444444   0.7222222
```

It can be quite fiddly to set up a complicated effort array so it may be easier to prepare it in advance as a .csv file and read it in, similar to how we read in the interaction matrix in Section 5.6. We give an example of this in Section 8.

7 Exploring the simulation results

In the previous sections we have seen how to set up a model and project it forward through time under our desired fishing scenario. The result of running a projection is an object of class *MizerSIm*. What do we then do? How can we explore the results of the simulation? In this section we introduce a range of summaries, plots and indicators that can be easily produced using methods included in `mizer`.

7.1 Summary methods for *MizerSim* objects

As well as the `summary()` methods that are available for both *MizerParams* and *MizerSim* objects, there are some useful summary methods to pull information out of a *MizerSim* object (see Table 3. All of these methods have help files to explain how they are used.

(It is also possible to use most of these methods with a *MizerParams* object if you also supply the population abundance as an argument. This can be useful for exploring how changes in parameter value or abundance can affect summary statistics and indicators. We won't explore this here but you can see their help files for more details.)

The methods `getBiomass()` and `getN()` have additional arguments that allow the user to set the size range over which to calculate the summary statistic. This is done by passing in a combination of the arguments `min_l`, `min_w`, `max_l` and `max_w` for the minimum and maximum length or weight. If `min_l` is specified there is no need to specify `min_w` and so on. However, if a length is specified (minimum or maximum) then it is necessary for the species parameter data.frame (see Section 5.1) to include the parameters a and b for length-weight conversion. It is possible to mix length and weight constraints, e.g. by supplying a minimum weight and a maximum length. The default values are the minimum and maximum weights of the spectrum.

7.1.1 Examples of using the summary methods

Here we show a simple demonstration of using a summary method. We first create a *MizerSim* object by projecting the parameters set up earlier through time. We then use `getSSB()` to calculate the SSB of each species through time.

```
> sim <- project(params,effort=0.5, t_max = 10, dt=1, t_save = 1)
> ssb <- getSSB(sim)
> head(ssb)
```

```
    sp
time      Sprat      Sandeel     N.pout      Herring          Dab     Whiting
   0    3492294      1049278    3474653      2914503 3.726664e+06     3900823
   1   56733931    112239822   34608419    546222486 2.993840e+09    71748514
   2 1837567494  10439182614 1041040825 130494268069 6.100689e+11  6027641191
   3 1547535335  45838022562 5464528013 471847479367 1.264044e+12 35568929854
   4  687113247  86772842776 6569714421 759417387620 1.287078e+12 64120369977
   5  275777810 122702111357 5510730650 871997258525 1.068019e+12 76202564981
    sp
time       Sole   Gurnard   Plaice     Haddock       Cod    Saithe
   0    2893657   3158689  3212613     3430207   4171128 4095286.7
   1  187512578  12087218  2311638    55642424   2749123 2699419.1
   2 1488455950 113186490  2310786  2183547524   1812913 1779339.5
   3 5188136091 478812559 11376522 15483556862  96963816 1179316.0
   4 7608628183 807095954 26117928 35485353839 420253798  896094.4
   5 8366358167 987168847 36457225 53339733940 844472104 1027845.6
```

As you can see, `getSSB()` calculates the SSB though time (depending on the argument t_save) for each species.

As mentioned above, we can specify the size range of some of the methods. For example, here we calculate the total biomass of each species but only include individuals that are longer than 10 cm and shorter than 50 cm.

```
> biomass <- getBiomass(sim, min_l = 10, max_l = 50)
> head(biomass)
```

```
    sp
time     Sprat      Sandeel      N.pout      Herring          Dab      Whiting
   0   4106144      3771472     4666727 4.574906e+06 5.033695e+06      5459982
   1  20983313    430802997    81811988 2.562692e+09 6.918006e+09    502479574
   2 633970322  40048558338  2534189427 6.119712e+11 1.407229e+12  42587876845
   3 831258645 154242929928 13130903462 1.703477e+12 2.209106e+12 107338738601
   4 517962687 245768188827 15521337041 2.019679e+12 1.948394e+12 137646354246
   5 258333022 318754494424 12826980160 1.931892e+12 1.489125e+12 138279587515
    sp
time        Sole    Gurnard    Plaice      Haddock          Cod      Saithe
   0     4958259    4945522   3184753      1788357     302806.3    303312.1
   1  1962695683  313376828  52332212   1719689167     874508.1    473576.0
   2 12032780013 2715078443 254144896  66181925319    2063652.1    790717.7
   3 22664750803 5418854700 418956854 243397408719  631365978.0  10432681.8
   4 24845161619 5792090986 407833781 374321729141 1082374986.5  51140634.2
   5 23018941521 5208128943 346328828 436368450385 1137714651.1 123638910.9
```

7.2 Methods for calculating indicators

Methods have been set up to calculate a range of indicators from a *MizerSim* object after a projection. These can be seen in Table 4. You can read the help pages for each of the methods for full instructions on how to use them, along with examples.

With all of the methods in the table it is possible to specify the size range of the community to be used in the calculation (e.g. to exclude very small or very large individuals) so that the calculated metrics can be compared to empirical data. This is done in the same way as the methods `getN()` and `getBiomass()` in Section 7.1. It is also possible to specify which species to include in the calculation. See the help files for more details.

7.2.1 Examples of calculating indicators

Here we show a simple demonstration of using indicator methods. We first create a *MizerSim* object by projecting the parameters set up earlier through time.

```
> sim <- project(params,effort=0.5, t_max = 10, dt=1, t_save = 1)
```

The slope of the community can be calculated using the `getCommunitySlope()` method. Initially we include all species and all sizes in the calculation (only the first five rows are shown):

```
> slope <- getCommunitySlope(sim)
> head(slope)
```

```
  time     slope intercept        r2
1    0 -1.521958  12.40786 0.9763343
2    1 -1.913260  17.47374 0.9043011
3    2 -2.204171  21.55067 0.8630065
4    3 -2.233434  23.11555 0.8927297
5    4 -2.155854  23.38581 0.9002788
6    5 -2.108532  23.51311 0.9049603
```

We can include only the species we want with the *species* argument. Here we only include demersal species. We also restrict the size range of the community that is used in the calculation to between 10 g and 5 kg.

```
> dem_species <- c("Dab","Whiting","Sole","Gurnard","Plaice","Haddock",
+     "Cod","Saithe")
> slope <- getCommunitySlope(sim, min_w = 10, max_w = 5000,
+     species = dem_species)
> head(head)

1 function (x, ...)
2 UseMethod("head")
```

7.3 Plotting the results

R is very powerful when it comes to exploring data through plots. A useful package for plotting is `ggplot2`. `ggplot2` uses data.frames for input data. Many of the summary methods and slots of the `mizer` classes are arrays or matrices. Fortunately it is straightforward to turn arrays and matrices into data.frames using the command `melt` which is in the `reshape` package. It is probably worth your time getting to grips with these packages to make plotting easier. PERHAPS INCLUDE EXAMPLE OF USING GGPLOT WITH MIZER BELOW?

Inlcuded in `mizer` are several dedicated plots that use *MizerSim* objects as inputs (see Table 5). As well as displaying the plots, these methods all return objects of type *ggplot* from the `ggplot2` package. See the help page of the plot methods for more details.

Some of the plots plot values by size (for example `plotFeedingLevel()` and `plotSpectra()`). For these plots, the default is to use the data at the final time step of the projection. With these plotting methods, it is also possible to specify a different time step to plot or a time range to average the values over before plotting.

7.3.1 Plotting examples

Using the plotting methods is straightforward. For example, to plot the total biomass of each species against time you use the `plotBiomass()` method:

```
> plotBiomass(sim)
```

The outcome of which can be seen in Figure 1.

As mentioned above, some of the plot methods plot values against size at a point in time (or averaged over a time period). For these plots it is possible to specify the time step to plot, or the time period to average the values over. The default is to use the final time step. Here we plot the abundance spectra (biomass), averaged over time = 5 to 10 (the simulation was projected until $t = 10$; see Figure 2):

```
> plotSpectra(sim, time_range = 5:10, biomass=TRUE)
```

8 A more detailed example using the North Sea

A long example.

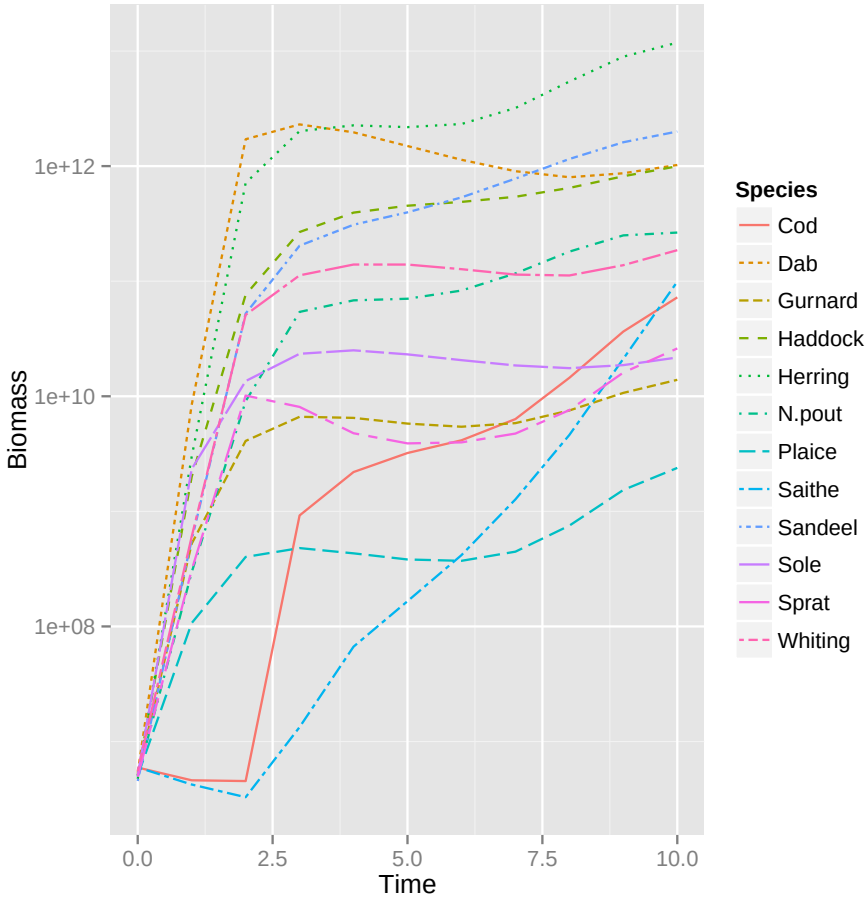

Figure 1: An example of using the `plotBiomass()` method.

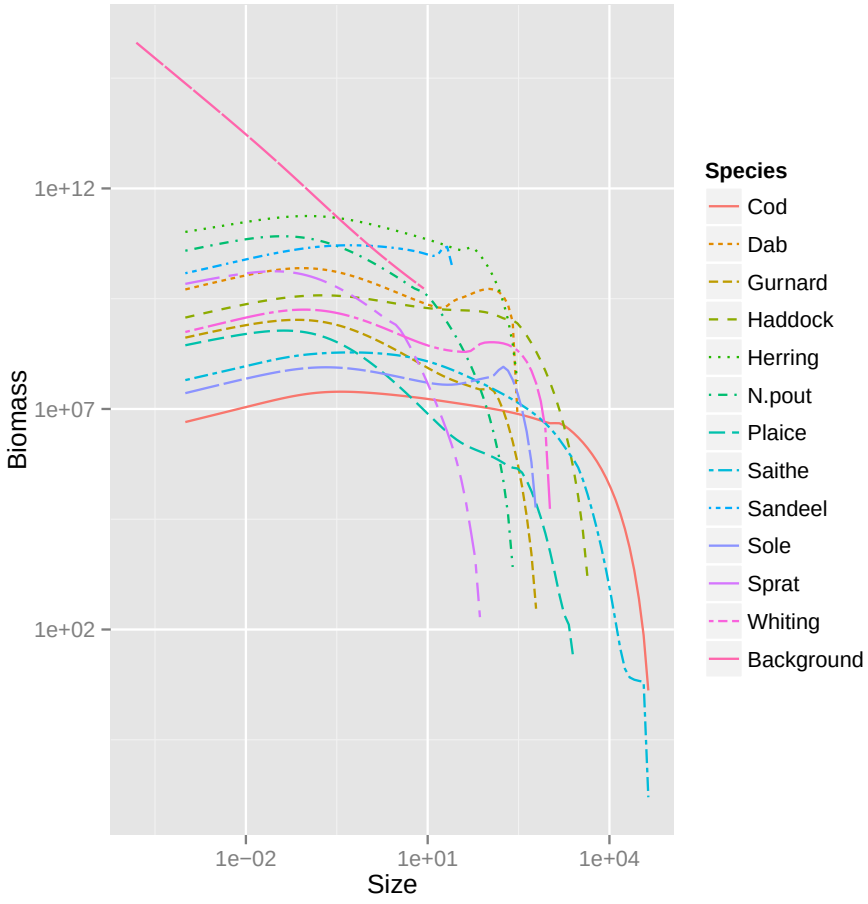

Figure 2: An example of using the `plotSpectra()` method, plotting values averaged over the period $t = 5$ to 10.

Run to equib at constant earliest fishing mortality on each species. Then historical fishing mortality on each species (same as the North Sea paper).

Compare outputs to empirical data.

9 Under the hood of `mizer`: notes for developers

Still to come.

Classes and whatnot

10 Acknowledgements

The authors of the `mizer` package would like to thank the authors of the `abc` package for allowing them to use some of the introductory text from their vignette.

Finlay Scott would like to thank Cefas Seedcorn Project DP266 for supporting this work.

References

K. H. Andersen and M. Pedersen. Damped trophic cascades driven by fishing in model marine ecosystems. *Proceedings of the Royal Society B-Biological Sciences*, 277(1682):795–802, March 2010. ISSN 0962-8452. doi: 10.1098/rspb.2009.1512. WOS:000273882800018.

Column name	Description	Default value
	Life history parameters	
species	Name of the species	Compulsory (no default)
w_inf	The asymptotic mass of the species	Compulsory (no default)
w_mat	Maturation mass. Used to calculate values for ψ. WHAT IS WMAT IN TERMS OF BIOLOGY? Is it the mass at first maturity?	Compulsory (no default)
h	Maximum food intake rate	Compulsory (no default)
gamma	Volumetric search rate	Compulsory (no default)
k	Activity coefficient	Compulsory (no default)
ks	Standard metabolism coefficient	Compulsory (no default)
beta	Preferred predator prey mass ratio	Compulsory (no default)
sigma	Width of prey size preference	Compulsory (no default)
z0	Background mortality (constant for all sizes)	Compulsory (no default)
alpha	Assimilation efficiency	0.6
erepro	Reproductive efficiency	1
w_min	The size class that recruits are placed in.	smallest size class of the species size spectrum
	Fishing gear parameters (see Section 5.2 for more details).	
sel_func	The name of the selectivity function to be used.	"sigmoid_length".
gear	The name of the fishing gear that selects the species. At the moment a species can only be selected by one gear.	Name of the species
catchability	The catchability of the fishing gear.	1
other columns	Other parameters used by the selectivity function. For example, if the default "sigmoid_length" function is used then the parameters "l25","l50","a" and "b" must also be specified as columns (see Section 5.2).	
	Stock recruitment parameters (see Section 5.3 for more details).	
other columns	Any arguments that appear in the stock-recruitment function must also have a column of values (see Section 5.3)	

Table 1: Columns of the species parameters data.frame

Argument	Description	Default value
min_w	The smallest size of the species community size spectrum. Note that this a different w_min to the one in the species parameter data.frame.	0.001
max_w	The largest size of the species size spectrum.	The laegest w_inf in the species parameters data.frame * 1.1
no_w	The number of size bins in the species size spectrum.	100
min_w_pp	The smallest size of the background size spectrum.	1e-10
no_w_pp	The number of size bins in the background size spectrum.	round(no_w) * 0.3
n	The scaling of intake.	2/3
p	The scaling of standard metabolism.	0.7
q	The search volume exponent.	0.8
r_pp	The growth rate of the primary productivity (the background spectrum).	10
kappa	The carrying capacity of the background spectrum.	1e11
lambda	The exponent of the background spectrum.	2+q-n
w_pp_cutoff	The cut off size of the background spectrum.	10

Table 2: Other parameters to the `MizerParams()` constructor

Method	Returns	Description
getSSB()	Two dimensional array (time x species)	Total Spawning Stock Biomass (SSB) of each species through time where SSB is calculated as EQUATION.
getBiomass()	Two dimensional array (time x species)	Total biomass of each species through time.
getN()	Two dimensional array (time x species)	Total abundance of each species through time.
getFeedingLevel()	Three dimensional array (time x species x size)	Feeding level of each species by size through time.
getFMort()	Three dimensional array (time x species x size)	Total fishing mortality on each species by size through time.
getFMortGear()	Four dimensional array (time x gear x species x size)	Fishing mortality on each species by each gear at size through time.
getYieldGear()	Three dimensional array (time x gear x species)	Total yield by gear and species through time.
getYield()	Two dimensional array (time x species)	Total yield of each species across all gears through time.

Table 3: Summary methods for *MizerSim* objects.

Method	Returns	Description
getProportionOfLargeFish()	A vector with values at each time step.	Calculates the proportion of large fish through time. The threshold value can be specified. It is possible to calculation the proportion of large fish based on either length or weight.
getMeanWeight()	A vector with values at each saved time step.	The mean weight of the community through time. This is calculated as the total biomass of the community divided by the total abundance.
getMeanMaxWeight()	Depends on the *measure* argument. If *measure* == *both* then you get a matrix with two columns, one with values by numbers, the other with values by biomass at each saved time step. If *measure* = *numbers* or *biomass* you get a vector of the respective values at each saved time step.	The mean maximum weight of the community through time. This can be calculated by numbers or by biomass. See the help file for more details.
getCommunitySlope()	A data.frame with four columns: time step, slope, intercept and R^2 value.	Calculates the slope of the community abundance spectrum through time by performing a linear regression on the logged total numerical abundance and logged body size.

Table 4: Indicator methods for *MizerSim* objects.

Plot	Description
plotBiomass()	Plots the total biomass of each species through time. The size range of the community can be specified in the same way as the method `getBiomass()`.
plotSpectra()	Plots the abundance (biomass or numbers) spectra of each species and the background community. It is possible to specify a minimum size which is useful for truncating the plot.
plotFeedingLevel()	Plots the feeding level of each species against size.
plotM2()	Plots the predation mortality of each species against size.
plotFMort()	Plots the total fishing mortality of each species against size.
plotYield()	Plots the total yield of each species across all fishing gears against time.
plotYieldGear()	Plots the total yield of each specis by gear against time.

Table 5: Plot methods for *MizerSim* objects.