# Package 'mizer'

July 28, 2020

**Title** Multi-Species sIZE Spectrum Modelling in R

**Type** Package

**Description** A set of classes and methods to set up and run multi-species, trait based and community size spectrum ecological models, focused on the marine environment.

**Maintainer** Gustav Delius <gustav.delius@york.ac.uk>

**Version** 2.0.3

**License** GPL-3

**Imports** assertthat, deSolve, dplyr, ggplot2, grid, methods, plotly, plyr, progress, Rcpp, reshape2,

**LinkingTo** Rcpp

**Depends** R (>= 3.1)

**Suggests** testthat (>= 2.1.0), vdiffr, roxygen2, knitr, shinyBS, rmarkdown, pkgdown, covr, spelling

**Collate** 'MizerParams-class.R' 'MizerSim-class.R' 'species_params.R' 'setInteraction.R' 'setPredKernel.R' 'setSearchVolume.R' 'setMaxIntakeRate.R' 'setMetabolicRate.R' 'setExtMort.R' 'setReproduction.R' 'setResource.R' 'setFishing.R' 'setInitialValues.R' 'upgrade.R' 'selectivity_funcs.R' 'pred_kernel_funcs.R' 'resource_dynamics.R' 'project.R' 'mizer-package.R' 'project_methods.R' 'rate_functions.R' 'summary_methods.R' 'plots.R' 'newMultispeciesParams.R' 'wrapper_functions.R' 'steady.R' 'extension.R' 'data.R' 'RcppExports.R' 'deprecated.R'

**RoxygenNote** 7.1.1

**Encoding** UTF-8

**LazyData** true

**URL** https://sizespectrum.org/mizer, https://github.com/sizespectrum/mizer

**BugReports** https://github.com/sizespectrum/mizer/issues

**Language** en-GB

**NeedsCompilation** yes

**Author** Gustav Delius [cre, aut, cph] (<https://orcid.org/0000-0003-4092-8228>),
Finlay Scott [aut, cph],
Julia Blanchard [aut, cph] (<https://orcid.org/0000-0003-0532-4824>),
Ken Andersen [aut, cph] (<https://orcid.org/0000-0002-8478-3430>),
Richard Southwell [ctb, cph]

**Repository** CRAN

**Date/Publication** 2020-07-28 14:30:02 UTC

# R topics documented:

---

`mizer-package` *mizer: Multi-species size-based modelling in R*

---

## Description

The mizer package implements multi-species size-based modelling in R. It has been designed for modelling marine ecosystems.

## Details

Using **mizer** is relatively simple. There are three main stages:

1. *Setting the model parameters*. This is done by creating an object of class MizerParams. This includes model parameters such as the life history parameters of each species, and the range of the size spectrum. There are several setup functions that help to create a MizerParams objects for particular types of models:
   - `newCommunityParams()`
   - `newTraitParams()`
   - `newMultispeciesParams()`
2. *Running a simulation*. This is done by calling the `project()` function with the model parameters. This produces an object of MizerSim that contains the results of the simulation.
3. *Exploring results*. After a simulation has been run, the results can be explored using a range of plotting_functions, summary_functions and indicator_functions.

See the mizer website for full details of the principles behind mizer and how the package can be used to perform size-based modelling.

## Author(s)

**Maintainer**: Gustav Delius <gustav.delius@york.ac.uk> (ORCID) [copyright holder]

Authors:

- Finlay Scott <drfinlayscott@gmail.com> [copyright holder]
- Julia Blanchard <julia.blanchard@utas.edu.au> (ORCID) [copyright holder]
- Ken Andersen <kha@aqua.dtu.dk> (ORCID) [copyright holder]

Other contributors:

- Richard Southwell <richard.southwell@york.ac.uk> [contributor, copyright holder]

## See Also

Useful links:

- https://sizespectrum.org/mizer
- https://github.com/sizespectrum/mizer
- Report bugs at https://github.com/sizespectrum/mizer/issues

---

`BevertonHoltRDD` *Beverton Holt function to calculate density-dependent reproduction rate*

---

## Description

Takes the density-independent rates $R_p$ of egg production and returns reduced, density-dependent reproduction rates $R$ given as

$$R = R_p \frac{R_{max}}{R_p + R_{max}}$$

where $R_{max}$ are the maximum possible reproduction rates that must be specified in a column in the species parameter dataframe.

## Usage

```
BevertonHoltRDD(rdi, species_params, ...)
```

## Arguments

| | |
|---|---|
| `rdi` | Vector of density-independent reproduction rates $R_p$ for all species. |
| `species_params` | A species parameter dataframe. Must contain a column R_max holding the maximum reproduction rate $R_{max}$ for each species. |
| `...` | Unused |

## Details

This is only one example of a density-dependence. You can write your own function based on this example, returning different density-dependent reproduction rates. Two other examples provided are `RickerRDD()` and `SheperdRDD()`. For more explanation see `setReproduction()`.

## Value

Vector of density-dependent reproduction rates.

## See Also

Other functions calculating density-dependent reproduction rate: `RickerRDD()`, `SheperdRDD()`, `constantRDD()`, `noRDD()`

## box_pred_kernel *Box predation kernel*

### Description

A predation kernel where the predator/prey mass ratio is uniformly distributed on an interval.

### Usage

```
box_pred_kernel(ppmr, ppmr_min, ppmr_max)
```

### Arguments

| | |
|---|---|
| ppmr | A vector of predator/prey size ratios |
| ppmr_min | Minimum predator/prey mass ratio |
| ppmr_max | Maximum predator/prey mass ratio |

### Details

Writing the predator mass as $w$ and the prey mass as $w_p$, the feeding kernel is 1 if $w/w_p$ is between ppmr_min and ppmr_max and zero otherwise. The parameters need to be given in the species parameter dataframe in the columns ppmr_min and ppmr_max.

### Value

A vector giving the value of the predation kernel at each of the predator/prey mass ratios in the ppmr argument.

## constantRDD *Give constant reproduction rate*

### Description

Simply returns the value from species_params$constant_reproduction.

### Usage

```
constantRDD(rdi, species_params, ...)
```

### Arguments

| | |
|---|---|
| rdi | Vector of density-independent reproduction rates $R_p$ for all species. |
| species_params | A species parameter dataframe. Must contain a column constant_reproduction. |
| ... | Unused |

## Value

Vector `species_params$constant_reproduction`

## See Also

Other functions calculating density-dependent reproduction rate: `BevertonHoltRDD()`, `RickerRDD()`, `SheperdRDD()`, `noRDD()`

---

`constant_other` *Helper function to keep other components constant*

---

## Description

Helper function to keep other components constant

## Usage

```
constant_other(params, n_other, component, ...)
```

## Arguments

| | |
|---|---|
| `params` | MizerParams object |
| `n_other` | Abundances of other components |
| `component` | Name of the component that is being updated |
| `...` | Unused |

---

`double_sigmoid_length` *Length based double-sigmoid selectivity function*

---

## Description

A hump-shaped selectivity function with a sigmoidal rise and an independent sigmoidal drop-off. This drop-off is what distinguishes this from the function `sigmoid_length()` and it is intended to model the escape of large individuals from the fishing gear.

## Usage

```
double_sigmoid_length(w, l25, l50, l50_right, l25_right, species_params, ...)
```

### Arguments

| | |
|---|---|
| w | the size of the individual. |
| l25 | the length which gives a selectivity of 25%. |
| l50 | the length which gives a selectivity of 50%. |
| l50_right | the length which gives a selectivity of 50%. |
| l25_right | the length which gives a selectivity of 25%. |
| species_params | A list with the species params for the current species. Used to get at the length-weight parameters a and b |
| ... | Unused |

### Details

The selectivity is obtained as the product of two sigmoidal curves, one rising and one dropping. The sigmoidal rise is based on the two parameters l25 and l50 which determine the length at which 25% and 50% of the stock is selected respectively. The sigmoidal drop-off is based on the two parameters l50_right and l25_right which determine the length at which the selectivity curve has dropped back to 50% and 25% respectively.

As the size-based model is weight based, and this selectivity function is length based, it uses the length-weight parameters a and b to convert between length and weight.

---

## emptyParams *Create empty MizerParams object of the right size*

---

### Description

An internal function. Sets up a valid MizerParams object with all the slots initialised and given dimension names, but with some slots left empty. This function is to be used by other functions to set up full parameter objects.

### Usage

```
emptyParams(
  species_params,
  gear_params = data.frame(),
  no_w = 100,
  min_w = 0.001,
  max_w = NA,
  min_w_pp = 1e-12
)
```

### Arguments

| | |
|---|---|
| species_params | A data frame of species-specific parameter values. |
| gear_params | A data frame with gear-specific parameter values. |
| no_w | The number of size bins in the consumer spectrum. |
| min_w | Sets the size of the eggs of all species for which this is not given in the w_min column of the species_params dataframe. |
| max_w | The largest size of the consumer spectrum. By default this is set to the largest w_inf specified in the species_params data frame. |
| min_w_pp | The smallest size of the resource spectrum. |

### Value

An empty but valid MizerParams object

### Size grid

A size grid is created so that the log-sizes are equally spaced. The spacing is chosen so that there will be no_w fish size bins, with the smallest starting at min_w and the largest starting at max_w. For w_full additional size bins are added below min_w, with the same log size. The number of extra bins is such that min_w_pp comes to lie within the smallest bin.

### Changes to species params

The species_params slot of the returned MizerParams object may differ slightly from the data frame supplied as argument to this function in the following ways:

- Default values are set for w_min,w_inf,alpha,gear,interaction_resource.
- The egg sizes in w_min are rounded down to lie on a grid point.

Note that the other characteristic sizes of the species, like w_mat and w_inf, are not modified to lie on grid points.

### See Also

See newMultispeciesParams() for a function that fills the slots left empty by this function.

---

## finalN *Size spectra at end of simulation*

---

### Description

Size spectra at end of simulation

## Usage

```
finalN(sim)

finalNResource(sim)
```

## Arguments

| | |
|---|---|
| sim | A MizerSim object |

## Value

For `finalN()`: An array (species x size) holding the consumer number densities at the end of the simulation

For `finalNResource()`: A vector holding the resource number densities at the end of the simulation for all size classes

---

`finalNOther` *Values of other ecosystem components at end of simulation*

---

## Description

Values of other ecosystem components at end of simulation

## Usage

```
finalNOther(sim)
```

## Arguments

| | |
|---|---|
| sim | A MizerSim object |

## Value

A named list holding the values of other ecosystem components at the end of the simulation

getBiomass *Calculate the total biomass of each species within a size range at each time step.*

## Description

Calculates the total biomass through time of the species in the MizerSim class within user defined size limits. The default option is to use the whole size range. You can specify minimum and maximum weight or length range for the species. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used).

## Usage

```
getBiomass(sim, ...)
```

## Arguments

| | |
|---|---|
| sim | An object of class MizerSim. |
| ... | Arguments passed on to get_size_range_array |
| | min_w Smallest weight in size range. Defaults to smallest weight in the model. |
| | max_w Largest weight in size range. Defaults to largest weight in the model. |
| | min_l Smallest length in size range. If supplied, this takes precedence over min_w. |
| | max_l Largest length in size range. If supplied, this takes precedence over max_w. |

## Value

An array containing the biomass (time x species)

## See Also

Other summary functions: getDiet(), getGrowthCurves(), getN(), getSSB(), getYieldGear(), getYield()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
getBiomass(sim)
getBiomass(sim, min_w = 10, max_w = 1000)

## End(Not run)
```

getBiomassFrame *Get data frame of biomass of species through time, ready for ggplot2*

## Description

After running a projection, the biomass of each species can be plotted against time. The biomass is calculated within user defined size limits (min_w, max_w, min_l, max_l, see `getBiomass()`).

## Usage

```
getBiomassFrame(
  sim,
  species = dimnames(sim@n)$sp[!is.na(sim@params@A)],
  start_time = as.numeric(dimnames(sim@n)[[1]][1]),
  end_time = as.numeric(dimnames(sim@n)[[1]][dim(sim@n)[1]]),
  ylim = c(NA, NA),
  total = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| sim | An object of class MizerSim |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| start_time | The first time to be plotted. Default is the beginning of the time series. |
| end_time | The last time to be plotted. Default is the end of the time series. |
| ylim | A numeric vector of length two providing lower and upper limits for the y axis. Use NA to refer to the existing minimum or maximum. Any values below 1e-20 are always cut off. |
| total | A boolean value that determines whether the total biomass from all species is plotted as well. Default is FALSE. |
| ... | Arguments passed on to `get_size_range_array`<br>`min_w` Smallest weight in size range. Defaults to smallest weight in the model.<br>`max_w` Largest weight in size range. Defaults to largest weight in the model.<br>`min_l` Smallest length in size range. If supplied, this takes precedence over `min_w`.<br>`max_l` Largest length in size range. If supplied, this takes precedence over `max_w`. |

## Value

A data frame

`getCommunitySlope` *Calculate the slope of the community abundance*

## Description

Calculates the slope of the community abundance through time by performing a linear regression on the logged total numerical abundance at weight and logged weights (natural logs, not log to base 10, are used). You can specify minimum and maximum weight or length range for the species. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used). You can also specify the species to be used in the calculation.

## Usage

```
getCommunitySlope(
  sim,
  species = seq_len(nrow(species_params(getParams(sim)))),
  biomass = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `sim` | A MizerSim object |
| `species` | Numeric or character vector of species to include in the calculation. |
| `biomass` | Boolean. If TRUE (default), the abundance is based on biomass, if FALSE the abundance is based on numbers. |
| `...` | Arguments passed on to `get_size_range_array`<br>`min_w` Smallest weight in size range. Defaults to smallest weight in the model.<br>`max_w` Largest weight in size range. Defaults to largest weight in the model.<br>`min_l` Smallest length in size range. If supplied, this takes precedence over `min_w`.<br>`max_l` Largest length in size range. If supplied, this takes precedence over `max_w`. |

## Value

A data.frame with four columns: time step, slope, intercept and the coefficient of determination R^2.

## See Also

Other functions for calculating indicators: `getMeanMaxWeight()`, `getMeanWeight()`, `getProportionOfLargeFish()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
sim <- project(params, effort=1, t_max=40, dt = 1, t_save = 1)
# Slope based on biomass, using all species and sizes
slope_biomass <- getCommunitySlope(sim)
# Slope based on numbers, using all species and sizes
slope_numbers <- getCommunitySlope(sim, biomass=FALSE)
# Slope based on biomass, using all species and sizes between 10g and 1000g
slope_biomass <- getCommunitySlope(sim, min_w = 10, max_w = 1000)
# Slope based on biomass, using only demersal species and sizes between 10g and 1000g
dem_species <- c("Dab","Whiting","Sole","Gurnard","Plaice","Haddock", "Cod","Saithe")
slope_biomass <- getCommunitySlope(sim, species = dem_species, min_w = 10, max_w = 1000)

## End(Not run)
```

---

# getComponent *Get information about other ecosystem components*

---

## Description

Get information about other ecosystem components

## Usage

```
getComponent(params, component)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| component | Name of the component of interest. If missing, a list of all components will be returned. |

## Value

A list with the entries `initial_value`, `dynamics_fun`, `encounter_fun`, `mort_fun`, `component_params`. If `component` is missing, then a list of lists for all components is returned.

## getCriticalFeedingLevel *Get critical feeding level*

### Description

The critical feeding level is the feeding level at which the food intake is just high enough to cover the metabolic costs, with nothing left over for growth or reproduction.

### Usage

```
getCriticalFeedingLevel(params)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |

### Value

A matrix (species x size) with the critical feeding level

## getDiet *Get diet of predator at size, resolved by prey species*

### Description

Calculates the rate at which a predator of a particular species and size consumes biomass of each prey species and resource.

### Usage

```
getDiet(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  proportion = TRUE
)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| proportion | If TRUE (default) the function returns the diet as a proportion of the total consumption rate. If FALSE it returns the consumption rate in grams. |

## Details

This function performs the same integration as `getEncounter()` but does not aggregate over prey species, and multiplies by (1-feeding_level) to get the consumed biomass rather than the available biomass. Outside the range of sizes for a predator species the returned rate is zero.

## Value

An array (predator species x predator size x (prey species + resource + other components) )

## See Also

Other summary functions: `getBiomass()`, `getGrowthCurves()`, `getN()`, `getSSB()`, `getYieldGear()`, `getYield()`

---

`getEffort` *Fishing effort used in simulation*

---

## Description

Note that the array returned may not be exactly the same as the `effort` argument that was passed in to `project()`. This is because only the saved effort is stored (the frequency of saving is determined by the argument `t_save`).

## Usage

```
getEffort(sim)
```

## Arguments

| | |
|---|---|
| `sim` | A MizerSim object |

## Value

An array (time x gear) that stores the fishing effort by time and gear.

getEGrowth *Get energy rate available for growth*

## Description

Calculates the energy rate $g_i(w)$ (grams/year) available by species and size for growth after metabolism, movement and reproduction have been accounted for.

## Usage

```
getEGrowth(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A two dimensional array (prey species x prey size)

## Your own growth rate function

By default `getEGrowth()` calls `mizerEGrowth()`. However you can replace this with your own alternative growth rate function. If your function is called `"myEGrowth"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "EGrowth", "myEGrowth")
```

Your function will then be called instead of `mizerEGrowth()`, with the same arguments.

## See Also

getERepro(), getEReproAndGrowth()

Other rate functions: getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getFeedingLevel(), getMort(), getPredMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the energy at a particular time step
getEGrowth(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ], t = 15)

## End(Not run)
```

---

# getEncounter *Get encounter rate*

---

## Description

Returns the rate at which a predator of species $i$ and weight $w$ encounters food (grams/year).

## Usage

```
getEncounter(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The current time. Unused |

## Value

A named two dimensional array (predator species x predator size) with the encounter rates.

### Predation encounter

The encounter rate $E_i(w)$ at which a predator of species $i$ and weight $w$ encounters food has contributions from the encounter of fish prey and of resource. This is determined by summing over all prey species and the resource spectrum and then integrating over all prey sizes $w_p$, weighted by predation kernel $\phi(w, w_p)$:

$$E_i(w) = \gamma_i(w) \int \left( \theta_{ip} N_R(w_p) + \sum_j \theta_{ij} N_j(w_p) \right) \phi_i(w, w_p) w_p \, dw_p.$$

Here $N_j(w)$ is the abundance density of species $j$ and $N_R(w)$ is the abundance density of resource. The overall prefactor $\gamma_i(w)$ determines the predation power of the predator. It could be interpreted as a search volume and is set with the `setSearchVolume()` function. The predation kernel $\phi(w, w_p)$ is set with the `setPredKernel()` function. The species interaction matrix $\theta_{ij}$ is set with `setInteraction()` and the resource interaction vector $\theta_{ip}$ is taken from the `interaction_resource` column in `params@species_params`.

### Details

The encounter rate is multiplied by $1 - f_0$ to obtain the consumption rate, where $f_0$ is the feeding level calculated with `getFeedingLevel()`. This is used by the `project()` function for performing simulations.

The function returns values also for sizes outside the size-range of the species. These values should not be used, as they are meaningless.

If your model contains additional components that you added with `setComponent()` and for which you specified an `encounter_fun` function then the encounters of these components will be included in the returned value.

### Your own encounter function

By default `getEncounter()` calls `mizerEncounter()`. However you can replace this with your own alternative encounter function. If your function is called `"myEncounter"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "Encounter", "myEncounter")
```

Your function will then be called instead of `mizerEncounter()`, with the same arguments.

### See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Run simulation with constant fishing effort for all gears for 20 years
```

```
sim <- project(params, t_max = 20, effort = 0.5)
getEncounter(params, n = finalN(sim), n_pp = finalNResource(sim), t = 20)

## End(Not run)
```

---

`getERepro` *Get energy rate available for reproduction*

---

## Description

Calculates the energy rate (grams/year) available for reproduction after growth and metabolism have been accounted for.

## Usage

```
getERepro(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `...` | Unused |

## Value

A two dimensional array (prey species x prey size) holding

$$\psi_i(w) E_{r.i}(w)$$

where $E_{r.i}(w)$ is the rate at which energy becomes available for growth and reproduction, calculated with `getEReproAndGrowth()`, and $\psi_i(w)$ is the proportion of this energy that is used for reproduction. This proportion is taken from the `params` object and is set with `setReproduction()`.

### Your own reproduction rate function

By default `getERepro()` calls `mizerERepro()`. However you can replace this with your own alternative reproduction rate function. If your function is called `"myERepro"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "ERepro", "myERepro")
```

Your function will then be called instead of `mizerERepro()`, with the same arguments.

### See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the energy at a particular time step
getERepro(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ], t = 15)

## End(Not run)
```

---

## getEReproAndGrowth *Get energy rate available for reproduction and growth*

---

### Description

Calculates the energy rate $E_{r.i}(w)$ (grams/year) available for reproduction and growth after metabolism and movement have been accounted for.

### Usage

```
getEReproAndGrowth(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A two dimensional array (species x size) holding

$$E_{r.i}(w) = \max(0, \alpha_i \, (1 - \texttt{feeding\_level}_i(w)) \, \texttt{encounter}_i(w) - \texttt{metab}_i(w)).$$

Due to the form of the feeding level, calculated by `getFeedingLevel()`, this can also be expressed as

$$E_{r.i}(w) = \max(0, \alpha_i \, \texttt{feeding\_level}_i(w) \, h_i(w) - \texttt{metab}_i(w))$$

where $h_i$ is the maximum intake rate, set with `setMaxIntakeRate()`. The assimilation rate $\alpha_i$ is taken from the species parameter data frame in `params`. The metabolic rate `metab` is taken from `params` and set with `setMetabolicRate()`.

The return value can be negative, which means that the energy intake does not cover the cost of metabolism and movement.

## Your own energy rate function

By default `getEReproAndGrowth()` calls `mizerEReproAndGrowth()`. However you can replace this with your own alternative energy rate function. If your function is called `"myEReproAndGrowth"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "EReproAndGrowth", "myEReproAndGrowth")
```

Your function will then be called instead of `mizerEReproAndGrowth()`, with the same arguments.

## See Also

The part of this energy rate that is invested into growth is calculated with `getEGrowth()` and the part that is invested into reproduction is calculated with `getERepro()`.

Other rate functions: `getEGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the energy at a particular time step
getEReproAndGrowth(params, n = N(sim)[15, , ], n_pp = NResource[15, ], t = 15)

## End(Not run)
```

---

`getESpawning` *Alias for getERepro*

---

## Description

An alias provided for backward compatibility with mizer version <= 1.0

## Usage

```
getESpawning(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `...` | Unused |

## Value

A two dimensional array (prey species x prey size) holding

$$\psi_i(w) E_{r.i}(w)$$

where $E_{r.i}(w)$ is the rate at which energy becomes available for growth and reproduction, calculated with `getEReproAndGrowth()`, and $\psi_i(w)$ is the proportion of this energy that is used for reproduction. This proportion is taken from the `params` object and is set with `setReproduction()`.

### Your own reproduction rate function

By default `getERepro()` calls `mizerERepro()`. However you can replace this with your own alternative reproduction rate function. If your function is called `"myERepro"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "ERepro", "myERepro")
```

Your function will then be called instead of `mizerERepro()`, with the same arguments.

## See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the energy at a particular time step
getERepro(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ], t = 15)

## End(Not run)
```

---

`getFeedingLevel` *Get feeding level*

---

## Description

Returns the feeding level. By default this function uses `mizerFeedingLevel()` to calculate the feeding level, but this can be overruled via `setRateFunction()`.

## Usage

```
getFeedingLevel(object, n, n_pp, n_other, time_range, drop = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `object` | A `MizerParams` object or a `MizerSim` object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `time_range` | A vector of times. Only the range of times is relevant, i.e., all times between the smallest and largest will be selected. The time_range can be character or numeric. |
| `drop` | If `TRUE` then any dimension of length 1 will be removed from the returned array. |
| `...` | Unused |

## Value

If a MizerParams object is passed in, the function returns a two dimensional array (predator species x predator size) based on the abundances also passed in. If a MizerSim object is passed in, the function returns a three dimensional array (time step x predator species x predator size) with the feeding level calculated at every time step in the simulation. If drop = TRUE then the dimension of length 1 will be removed from the returned array.

## Feeding level

The feeding level $f_i(w)$ is the proportion of its maximum intake rate at which the predator is actually taking in fish. It is calculated from the encounter rate $E_i$ and the maximum intake rate $h_i(w)$ as

$$f_i(w) = \frac{E_i(w)}{E_i(w) + h_i(w)}.$$

The encounter rate $E_i$ is passed as an argument or calculated with getEncounter(). The maximum intake rate $h_i(w)$ is taken from the params object, and is set with setMaxIntakeRate(). As a consequence of the above expression for the feeding level, $1 - f_i(w)$ is the proportion of the food available to it that the predator actually consumes.

## Your own feeding level function

By default getFeedingLevel() calls mizerFeedingLevel(). However you can replace this with your own alternative feeding level function. If your function is called "myFeedingLevel" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "FeedingLevel", "myFeedingLevel")
```

Your function will then be called instead of mizerFeedingLevel(), with the same arguments.

## See Also

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getMort(), getPredMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Get initial feeding level
fl <- getFeedingLevel(params)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the feeding level at all saved time steps
fl <- getFeedingLevel(sim)
# Get the feeding level for years 15 - 20
fl <- getFeedingLevel(sim, time_range = c(15, 20))

## End(Not run)
```

getFMort *Get the total fishing mortality rate from all fishing gears by time, species and size.*

## Description

Calculates the total fishing mortality (in units 1/year) from all gears by species and size at each time step in the `effort` argument. The total fishing mortality is just the sum of the fishing mortalities imposed by each gear, $\mu_{f.i}(w) = \sum_g F_{g,i,w}$.

## Usage

```
getFMort(object, effort, time_range, drop = TRUE)
```

## Arguments

| | |
|---|---|
| `object` | A `MizerParams` object or a `MizerSim` object |
| `effort` | The effort of each fishing gear. Only used if the object argument is of class `MizerParams`. See notes below. |
| `time_range` | Subset the returned fishing mortalities by time. The time range is either a vector of values, a vector of min and max time, or a single value. Default is the whole time range. Only used if the `object` argument is of type `MizerSim`. |
| `drop` | Only used when object is of type `MizerSim`. Should dimensions of length 1 be dropped, e.g. if your community only has one species it might make presentation of results easier. Default is TRUE |

## Value

An array. If the effort argument has a time dimension, or object is of class `MizerSim`, the output array has three dimensions (time x species x size). If the effort argument does not have a time dimension, the output array has two dimensions (species x size).

## Note

Here: fishing mortality = catchability x selectivity x effort.

The `effort` argument is only used if a `MizerParams` object is passed in. The `effort` argument can be a two dimensional array (time x gear), a vector of length equal to the number of gears (each gear has a different effort that is constant in time), or a single numeric value (each gear has the same effort that is constant in time). The order of gears in the `effort` argument must be the same as in the `MizerParams` object.

If the object argument is of class `MizerSim` then the effort slot of the `MizerSim` object is used and the `effort` argument is not used.

## See Also

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFeedingLevel(), getMort(), getPredMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Get the total fishing mortality when effort is constant for all
# gears and time:
getFMort(params, effort = 1)
# Get the total fishing mortality when effort is different
# between the four gears but constant in time:
getFMort(params, effort = c(0.5,1,1.5,0.75))
# Get the total fishing mortality when effort is different
# between the four gears and changes with time:
effort <- array(NA, dim = c(20,4))
effort[, 1] <- seq(from = 0, to = 1, length = 20)
effort[, 2] <- seq(from = 1, to = 0.5, length = 20)
effort[, 3] <- seq(from = 1, to = 2, length = 20)
effort[, 4] <- seq(from = 2, to = 1, length = 20)
getFMort(params, effort = effort)
# Get the total fishing mortality using the effort already held in a
# MizerSim object.
sim <- project(params, t_max = 20, effort = 0.5)
getFMort(sim)
getFMort(sim, time_range = c(10, 20))

## End(Not run)
```

---

# getFMortGear *Get the fishing mortality by time, gear, species and size*

---

## Description

Calculates the fishing mortality rate $F_{g,i,w}$ by gear, species and size at each time step in the `effort` argument (in units 1/year).

## Usage

```
getFMortGear(object, effort, time_range)
```

## Arguments

| | |
|---|---|
| `object` | A `MizerParams` object or a `MizerSim` object. |
| `effort` | The effort for each fishing gear. See notes below. |
| `time_range` | Subset the returned fishing mortalities by time. The time range is either a vector of values, a vector of min and max time, or a single value. Default is the whole time range. Only used if the `object` argument is of type `MizerSim`. |

## Value

An array. If the effort argument has a time dimension, or a MizerSim is passed in, the output array has four dimensions (time x gear x species x size). If the effort argument does not have a time dimension (i.e. it is a vector or a single numeric), the output array has three dimensions (gear x species x size).

## Note

Here: fishing mortality = catchability x selectivity x effort.

The effort argument is only used if a MizerParams object is passed in. The effort argument can be a two dimensional array (time x gear), a vector of length equal to the number of gears (each gear has a different effort that is constant in time), or a single numeric value (each gear has the same effort that is constant in time). The order of gears in the effort argument must be the same the same as in the MizerParams object. If the effort argument is not supplied, its value is taken from the @initial_effort slot in the params object.

If the object argument is of class MizerSim then the effort slot of the MizerSim object is used and the effort argument is not used.

## See Also

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMort(), getFeedingLevel(), getMort(), getPredMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Get the fishing mortality when effort is constant
# for all gears and time:
getFMortGear(params, effort = 1)
# Get the fishing mortality when effort is different
# between the four gears but constant in time:
getFMortGear(params, effort = c(0.5, 1, 1.5, 0.75))
# Get the fishing mortality when effort is different
# between the four gears and changes with time:
effort <- array(NA, dim = c(20, 4))
effort[, 1] <- seq(from=0, to = 1, length = 20)
effort[, 2] <- seq(from=1, to = 0.5, length = 20)
effort[, 3] <- seq(from=1, to = 2, length = 20)
effort[, 4] <- seq(from=2, to = 1, length = 20)
getFMortGear(params, effort = effort)
# Get the fishing mortality using the effort already held in a MizerSim object.
sim <- project(params, t_max = 20, effort = 0.5)
getFMortGear(sim)
getFMortGear(sim, time_range = c(10, 20))

## End(Not run)
```

`getGrowthCurves` *Get growth curves giving weight as a function of age*

## Description

Get growth curves giving weight as a function of age

## Usage

```
getGrowthCurves(object, species, max_age = 20, percentage = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | MizerSim or MizerParams object. If given a MizerSim object, uses the growth rates at the final time of a simulation to calculate the size at age. If given a MizerParams object, uses the initial growth rates instead. |
| `species` | Name or vector of names of the species to be included. By default all species are included. |
| `max_age` | The age up to which to run the growth curve. Default is 20. |
| `percentage` | Boolean value. If TRUE, the size is given as a percentage of the maximal size. |

## Value

An array (species x age) containing the weight in grams.

## See Also

Other summary functions: `getBiomass()`, `getDiet()`, `getN()`, `getSSB()`, `getYieldGear()`, `getYield()`

## Examples

```
## Not run:
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
getGrowthCurves(params)
sim <- project(params, effort=1, t_max = 20, t_save = 2, progress_bar = FALSE)
getGrowthCurves(sim, max_age = 24)

## End(Not run)
```

getM2 *Alias for getPredMort*

## Description

An alias provided for backward compatibility with mizer version <= 1.0

## Usage

```
getM2(object, n, n_pp, n_other, time_range, drop = TRUE, ...)
```

## Arguments

| | |
|---|---|
| object | A MizerParams object or a MizerSim object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| time_range | A vector of times. Only the range of times is relevant, i.e., all times between the smallest and largest will be selected. The time_range can be character or numeric. |
| drop | If TRUE then any dimension of length 1 will be removed from the returned array. |
| ... | Unused |

## Value

If a MizerParams object is passed in, the function returns a two dimensional array (prey species x prey size) based on the abundances also passed in. If a MizerSim object is passed in, the function returns a three dimensional array (time step x prey species x prey size) with the predation mortality calculated at every time step in the simulation. Dimensions may be dropped if they have length 1 unless drop = FALSE.

## Your own predation mortality function

By default getPredMort() calls mizerPredMort(). However you can replace this with your own alternative predation mortality function. If your function is called "myPredMort" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "PredMort", "myPredMort")
```

Your function will then be called instead of mizerPredMort(), with the same arguments.

## See Also

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getFeedingLevel(), getMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get predation mortality at one time step
getPredMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ])
# Get predation mortality at all saved time steps
getPredMort(sim)
# Get predation mortality over the years 15 - 20
getPredMort(sim, time_range = c(15, 20))

## End(Not run)
```

---

# getM2Background *Alias for getResourceMort*

---

## Description

An alias provided for backward compatibility with mizer version <= 1.0

## Usage

```
getM2Background(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A vector of mortality rate by resource size.

### Your own resource mortality function

By default `getResourceMort()` calls `mizerResourceMort()`. However you can replace this with your own alternative resource mortality function. If your function is called `"myResourceMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "ResourceMort", "myResourceMort")
```

Your function will then be called instead of `mizerResourceMort()`, with the same arguments.

## See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get resource mortality at one time step
getResourceMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ])

## End(Not run)
```

---

# getMeanMaxWeight *Calculate the mean maximum weight of the community*

---

## Description

Calculates the mean maximum weight of the community through time. This can be calculated by numbers or biomass. The calculation is the sum of the w_inf * abundance of each species, divided by the total abundance community, where abundance is either in biomass or numbers. You can specify minimum and maximum weight or length range for the species. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used). You can also specify the species to be used in the calculation.

## Usage

```
getMeanMaxWeight(
  sim,
  species = seq_len(nrow(species_params(getParams(sim)))),
  measure = "both",
  ...
)
```

### Arguments

| | |
|---|---|
| sim | A MizerSim object |
| species | Numeric or character vector of species to include in the calculation. |
| measure | The measure to return. Can be 'numbers', 'biomass' or 'both' |
| ... | Arguments passed on to get_size_range_array |
| | min_w Smallest weight in size range. Defaults to smallest weight in the model. |
| | max_w Largest weight in size range. Defaults to largest weight in the model. |
| | min_l Smallest length in size range. If supplied, this takes precedence over min_w. |
| | max_l Largest length in size range. If supplied, this takes precedence over max_w. |

### Value

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.

### See Also

Other functions for calculating indicators: getCommunitySlope(), getMeanWeight(), getProportionOfLargeFish()

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
sim <- project(params, effort=1, t_max=10)
getMeanMaxWeight(sim)
getMeanMaxWeight(sim, species=c("Herring","Sprat","N.pout"))
getMeanMaxWeight(sim, min_w = 10, max_w = 5000)

## End(Not run)
```

---

## getMeanWeight *Calculate the mean weight of the community*

---

### Description

Calculates the mean weight of the community through time. This is simply the total biomass of the community divided by the abundance in numbers. You can specify minimum and maximum weight or length range for the species. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used). You can also specify the species to be used in the calculation.

## Usage

```
getMeanWeight(
  sim,
  species = seq_len(nrow(species_params(getParams(sim)))),
  ...
)
```

## Arguments

| | |
|---|---|
| sim | A MizerSim object |
| species | Numeric or character vector of species to include in the calculation. |
| ... | Arguments passed on to get_size_range_array<br>min_w Smallest weight in size range. Defaults to smallest weight in the model.<br>max_w Largest weight in size range. Defaults to largest weight in the model.<br>min_l Smallest length in size range. If supplied, this takes precedence over min_w.<br>max_l Largest length in size range. If supplied, this takes precedence over max_w. |

## Value

A vector containing the mean weight of the community through time

## See Also

Other functions for calculating indicators: getCommunitySlope(), getMeanMaxWeight(), getProportionOfLargeFish()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
sim <- project(params, effort=1, t_max=10)
getMeanWeight(sim)
getMeanWeight(sim, species=c("Herring","Sprat","N.pout"))
getMeanWeight(sim, min_w = 10, max_w = 5000)

## End(Not run)
```

---

# getMort *Get total mortality rate*

---

## Description

Calculates the total mortality rate $\mu_i(w)$ (in units 1/year) on each species by size from predation mortality, background mortality and fishing mortality for a single time step.

## Usage

```
getMort(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  effort = getInitialEffort(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| effort | A numeric vector of the effort by gear or a single numeric effort value which is used for all gears. |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Details

If your model contains additional components that you added with `setComponent()` and for which you specified a `mort_fun` function then the mortality inflicted by these components will be included in the returned value.

## Value

A two dimensional array (prey species x prey size).

## Your own mortality function

By default `getMort()` calls `mizerMort()`. However you can replace this with your own alternative mortality function. If your function is called `"myMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "Mort", "myMort")
```

Your function will then be called instead of `mizerMort()`, with the same arguments.

## See Also

`getPredMort()`, `getFMort()`

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the total mortality at a particular time step
getMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ],
        t = 15, effort = 0.5)

## End(Not run)
```

---

`getN` *Calculate the number of individuals within a size range*

---

## Description

Calculates the number of individuals within user-defined size limits, for each time and each species in the `MizerSim` object. The default option is to use the whole size range. You can specify minimum and maximum weight or lengths for the species. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used)

## Usage

```
getN(sim, ...)
```

## Arguments

| | |
|---|---|
| `sim` | An object of class `MizerSim`. |
| `...` | Arguments passed on to `get_size_range_array` |
| | `min_w` Smallest weight in size range. Defaults to smallest weight in the model. |
| | `max_w` Largest weight in size range. Defaults to largest weight in the model. |
| | `min_l` Smallest length in size range. If supplied, this takes precedence over `min_w`. |
| | `max_l` Largest length in size range. If supplied, this takes precedence over `max_w`. |

## Value

An array containing the total numbers (time x species)

## See Also

Other summary functions: `getBiomass()`, `getDiet()`, `getGrowthCurves()`, `getSSB()`, `getYieldGear()`, `getYield()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
getN(sim)
getN(sim, min_w = 10, max_w = 1000)

## End(Not run)
```

---

`getParams` *Extract the parameter object underlying a simulation*

---

## Description

Extract the parameter object underlying a simulation

## Usage

```
getParams(sim)
```

## Arguments

| | |
|---|---|
| `sim` | A MizerSim object |

## Value

The MizerParams object that was used to run the simulation

---

`getPhiPrey` *Get available energy*

---

## Description

This is deprecated and is no longer used by the mizer project() method. Calculates the amount $E_{a,i}(w)$ of food exposed to each predator as a function of predator size.

## Usage

```
getPhiPrey(object, n, n_pp, ...)
```

## Arguments

| | |
|---|---|
| `object` | An MizerParams object |
| `n` | A matrix of species abundances (species x size) |
| `n_pp` | A vector of the background abundance by size |
| `...` | Other arguments (currently unused) |

## Value

A two dimensional array (predator species x predator size)

## See Also

project()

## Examples

```
## Not run:
data(NS_species_params_gears)
data(inter)
params <- MizerParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
n <- sim@n[21,,]
n_pp <- sim@n_pp[21,]
getPhiPrey(params,n,n_pp)

## End(Not run)
```

---

# getPredKernel *Get predation kernel*

---

## Description

If no explicit predation kernel $\phi_i(w, w_p)$ is stored in the params object, then this function calculates it from the information in the species parameter data frame in the params object.

## Usage

```
getPredKernel(params)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |

## Details

For more detail about the predation kernel see setPredKernel().

## Value

An array (predator species x predator_size x prey_size)

getPredMort *Get total predation mortality rate*

## Description

Calculates the total predation mortality rate $\mu_{p,i}(w_p)$ (in units of 1/year) on each prey species by prey size:

$$\mu_{p.i}(w_p) = \sum_j \texttt{pred\_rate}_j(w_p)\, \theta_{ji}.$$

## Usage

```
getPredMort(object, n, n_pp, n_other, time_range, drop = TRUE, ...)
```

## Arguments

| | |
|---|---|
| object | A MizerParams object or a MizerSim object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| time_range | A vector of times. Only the range of times is relevant, i.e., all times between the smallest and largest will be selected. The time_range can be character or numeric. |
| drop | If TRUE then any dimension of length 1 will be removed from the returned array. |
| ... | Unused |

## Value

If a MizerParams object is passed in, the function returns a two dimensional array (prey species x prey size) based on the abundances also passed in. If a MizerSim object is passed in, the function returns a three dimensional array (time step x prey species x prey size) with the predation mortality calculated at every time step in the simulation. Dimensions may be dropped if they have length 1 unless drop = FALSE.

## Your own predation mortality function

By default getPredMort() calls mizerPredMort(). However you can replace this with your own alternative predation mortality function. If your function is called "myPredMort" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "PredMort", "myPredMort")
```

Your function will then be called instead of mizerPredMort(), with the same arguments.

## See Also

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getFeedingLevel(), getMort(), getPredRate(), getRDD(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get predation mortality at one time step
getPredMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ])
# Get predation mortality at all saved time steps
getPredMort(sim)
# Get predation mortality over the years 15 - 20
getPredMort(sim, time_range = c(15, 20))

## End(Not run)
```

---

# getPredRate *Get predation rate*

---

## Description

Calculates the potential rate (in units 1/year) at which a prey individual of a given size $w$ is killed by predators from species $j$. In formulas

$$\texttt{pred\_rate}_j(w_p) = \int \phi_j(w, w_p)(1 - f_j(w))\gamma_j(w)N_j(w)\,dw.$$

This potential rate is used in getPredMort() to calculate the realised predation mortality rate on the prey individual.

## Usage

```
getPredRate(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A two dimensional array (predator species x prey size), where the prey size runs over fish community plus resource spectrum.

## Your own predation rate function

By default `getPredRate()` calls `mizerPredRate()`. However you can replace this with your own alternative predation rate function. If your function is called `"myPredRate"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "PredRate", "myPredRate")
```

Your function will then be called instead of `mizerPredRate()`, with the same arguments.

## See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getRDD()`, `getRDI()`, `getResourceMort()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the feeding level at one time step
getPredRate(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ])

## End(Not run)
```

getProportionOfLargeFish

*Calculate the proportion of large fish*

## Description

Calculates the proportion of large fish through time in the MizerSim class within user defined size limits. The default option is to use the whole size range. You can specify minimum and maximum size ranges for the species and also the threshold size for large fish. Sizes can be expressed as weight or size. Lengths take precedence over weights (i.e. if both min_l and min_w are supplied, only min_l will be used). You can also specify the species to be used in the calculation. This function can be used to calculate the Large Fish Index. The proportion is based on either abundance or biomass.

## Usage

```
getProportionOfLargeFish(
  sim,
  species = seq_len(nrow(species_params(getParams(sim)))),
  threshold_w = 100,
  threshold_l = NULL,
  biomass_proportion = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| sim | A MizerSim object |
| species | Numeric or character vector of species to include in the calculation. |
| threshold_w | the size used as the cutoff between large and small fish. Default value is 100. |
| threshold_l | the size used as the cutoff between large and small fish. |
| biomass_proportion | a boolean value. If TRUE the proportion calculated is based on biomass, if FALSE it is based on numbers of individuals. Default is TRUE. |
| ... | Arguments passed on to get_size_range_array<br>min_w Smallest weight in size range. Defaults to smallest weight in the model.<br>max_w Largest weight in size range. Defaults to largest weight in the model.<br>min_l Smallest length in size range. If supplied, this takes precedence over min_w.<br>max_l Largest length in size range. If supplied, this takes precedence over max_w. |

## Value

A vector containing the proportion of large fish through time

## See Also

Other functions for calculating indicators: `getCommunitySlope()`, `getMeanMaxWeight()`, `getMeanWeight()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
sim <- project(params, effort=1, t_max=10)
getProportionOfLargeFish(sim)
getProportionOfLargeFish(sim, species=c("Herring","Sprat","N.pout"))
getProportionOfLargeFish(sim, min_w = 10, max_w = 5000)
getProportionOfLargeFish(sim, min_w = 10, max_w = 5000, threshold_w = 500)
getProportionOfLargeFish(sim, min_w = 10, max_w = 5000,
    threshold_w = 500, biomass_proportion=FALSE)

## End(Not run)
```

---

# getRDD *Get density dependent reproduction rate*

---

## Description

Calculates the density dependent rate of egg production $R_i$ (units 1/year) for each species. This is the flux entering the smallest size class of each species. The density dependent rate is the density independent rate obtained with `getRDI()` after it has been put through the density dependence function. This is the Beverton-Holt function `BevertonHoltRDD()` by default, but this can be changed. See `setReproduction()` for more details.

## Usage

```
getRDD(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  rdi = getRDI(params, n = n, n_pp = n_pp, n_other = n_other, t = t)
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |

rdi — A vector of density-independent reproduction rates for each species. If not specified, rdi is calculated internally using getRDI().

## Value

A numeric vector the length of the number of species.

## See Also

getRDI()

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getFeedingLevel(), getMort(), getPredMort(), getPredRate(), getRDI(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the rate at a particular time step
getRDD(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ], t = 15)

## End(Not run)
```

---

# getRDI — *Get density independent rate of egg production*

---

## Description

Calculates the density independent rate of egg production $R_{p.i}$ (units 1/year) before density dependence, by species. Used by getRDD() to calculate the actual density dependent rate. See setReproduction() for more details.

## Usage

```
getRDI(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A numeric vector the length of the number of species

## See Also

getRDD()

Other rate functions: getEGrowth(), getEReproAndGrowth(), getERepro(), getEncounter(), getFMortGear(), getFMort(), getFeedingLevel(), getMort(), getPredMort(), getPredRate(), getRDD(), getResourceMort()

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the density-independent reproduction rate at a particular time step
getRDI(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ], t = 15)

## End(Not run)
```

---

# getResourceMort *Get predation mortality rate for resource*

---

## Description

Calculates the predation mortality rate $\mu_p(w)$ on the resource spectrum by resource size (in units 1/year).

## Usage

```
getResourceMort(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Value

A vector of mortality rate by resource size.

## Your own resource mortality function

By default `getResourceMort()` calls `mizerResourceMort()`. However you can replace this with your own alternative resource mortality function. If your function is called `"myResourceMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "ResourceMort", "myResourceMort")
```

Your function will then be called instead of `mizerResourceMort()`, with the same arguments.

## See Also

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getMort()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get resource mortality at one time step
getResourceMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ])

## End(Not run)
```

## getSSB *Calculate the SSB of species*

### Description

Calculates the spawning stock biomass (SSB) through time of the species in the `MizerSim` class. SSB is calculated as the total mass of all mature individuals.

### Usage

```
getSSB(sim)
```

### Arguments

| | |
|---|---|
| `sim` | An object of class `MizerSim`. |

### Value

An array containing the SSB (time x species)

### See Also

Other summary functions: `getBiomass()`, `getDiet()`, `getGrowthCurves()`, `getN()`, `getYieldGear()`, `getYield()`

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
getSSB(sim)

## End(Not run)
```

## getTimes *Times for which simulation results are available*

### Description

Times for which simulation results are available

### Usage

```
getTimes(sim)
```

### Arguments

| | |
|---|---|
| sim | A MizerSim object |

### Value

A numeric vectors of the times (in years) at which simulation results have been stored in the MizerSim object.

---

## getYield *Calculate the total yield of each species*

---

### Description

Calculates the total yield of each species across all gears at each simulation time step.

### Usage

```
getYield(sim)
```

### Arguments

| | |
|---|---|
| sim | An object of class MizerSim. |

### Value

An array containing the total yield (time x species)

### See Also

getYieldGear()

Other summary functions: getBiomass(), getDiet(), getGrowthCurves(), getN(), getSSB(), getYieldGear()

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
sim <- project(params, effort=1, t_max=10)
y <- getYield(sim)

## End(Not run)
```

## getYieldGear *Calculate the total yield per gear and species*

### Description

Calculates the total yield per gear and species at each simulation time step.

### Usage

```
getYieldGear(sim)
```

### Arguments

| | |
|---|---|
| sim | An object of class MizerSim. |

### Value

An array containing the total yield (time x gear x species)

### See Also

getYield()

Other summary functions: getBiomass(), getDiet(), getGrowthCurves(), getN(), getSSB(), getYield()

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
getYieldGear(sim)

## End(Not run)
```

## getZ *Alias for getMort*

### Description

An alias provided for backward compatibility with mizer version <= 1.0

## Usage

```
getZ(
  params,
  n = initialN(params),
  n_pp = initialNResource(params),
  n_other = initialNOther(params),
  effort = getInitialEffort(params),
  t = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| effort | A numeric vector of the effort by gear or a single numeric effort value which is used for all gears. |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| ... | Unused |

## Details

If your model contains additional components that you added with `setComponent()` and for which you specified a `mort_fun` function then the mortality inflicted by these components will be included in the returned value.

## Value

A two dimensional array (prey species x prey size).

## Your own mortality function

By default `getMort()` calls `mizerMort()`. However you can replace this with your own alternative mortality function. If your function is called `"myMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "Mort", "myMort")
```

Your function will then be called instead of `mizerMort()`, with the same arguments.

## See Also

`getPredMort()`, `getFMort()`

Other rate functions: `getEGrowth()`, `getEReproAndGrowth()`, `getERepro()`, `getEncounter()`, `getFMortGear()`, `getFMort()`, `getFeedingLevel()`, `getPredMort()`, `getPredRate()`, `getRDD()`, `getRDI()`, `getResourceMort()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)
# Project with constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# Get the total mortality at a particular time step
getMort(params, n = N(sim)[15, , ], n_pp = NResource(sim)[15, ],
        t = 15, effort = 0.5)

## End(Not run)
```

---

`get_initial_n` *Calculate initial population abundances for the community populations*

---

## Description

This function uses the model parameters and other parameters to calculate initial population abundances for the community populations. These initial abundances should be reasonable guesses at the equilibrium values. The returned population can be passed to the `project` function.

## Usage

```
get_initial_n(params, n0_mult = NULL, a = 0.35)
```

## Arguments

| | |
|---|---|
| `params` | The model parameters. An object of type MizerParams. |
| `n0_mult` | Multiplier for the abundance at size 0. Default value is kappa/1000. |
| `a` | A parameter with a default value of 0.35. |

## Value

A matrix (species x size) of population abundances.

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears)
init_n <- get_initial_n(params)

## End(Not run)
```

## get_required_reproduction *Determine reproduction rate needed for initial egg abundance*

### Description

Determine reproduction rate needed for initial egg abundance

### Usage

```
get_required_reproduction(params)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |

### Value

A vector of reproduction rates for all species

## idxFinalT *Time index at end of simulation*

### Description

Time index at end of simulation

### Usage

```
idxFinalT(sim)
```

### Arguments

| | |
|---|---|
| sim | A MizerSim object |

### Value

An integer giving the index for extracting the results for the final time step

## Examples

```
## Not run:
sim <- project(NS_params, t_max = 12, t_save = 0.5)
idx <- idxFinalT(sim)
idx
# This coincides with
length(getTimes(sim))
# and corresponds to the final time
getTimes(sim)[idx]
# We can use this index to extract the result at the final time
identical(N(sim)[idx, , ], finalN(sim))
identical(NResource(sim)[idx, ], finalNResource(sim))

## End(Not run)
```

---

# indicator_functions *Description of indicator functions*

---

## Description

Mizer provides a range of functions to calculate indicators from a MizerSim object.

## Details

A list of available indicator functions for MizerSim objects is given in the table below

| Function | Returns |
|---|---|
| getProportionOfLargeFish() | A vector with values at each time step. |
| getMeanWeight() | A vector with values at each saved time step. |
| getMeanMaxWeight() | Depends on the measure argument. If measure = "both" then you get a matrix with two col |
| getCommunitySlope() | A data.frame with four columns: time step, slope, intercept and the coefficient of determina |

## See Also

summary_functions, plotting_functions

---

# initialN<- *Initial values for fish spectra*

---

## Description

Values used as starting values for simulations with `project()`.

### Usage

```
initialN(params) <- value

initialN(object)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| value | A matrix with dimensions species x size holding the initial number densities for the fish spectra. |
| object | An object of class MizerParams or MizerSim |

---

## initialNOther<- *Initial values for other ecosystem components*

---

### Description

Values used as starting values for simulations with `project()`.

### Usage

```
initialNOther(params) <- value

initialNOther(params)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| value | A named list with the initial values of other ecosystem components |

---

## initialNResource<- *Initial value for resource spectrum*

---

### Description

Value used as starting value for simulations with `project()`.

### Usage

```
initialNResource(params) <- value

initialNResource(object)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| value | A vector with the initial number densities for the resource spectrum |
| object | An object of class MizerParams or MizerSim |

---

## inter *Example interaction matrix for the North Sea example*

---

### Description

The interaction coefficient between predators and preys in the North Sea.

### Format

A 12 x 12 matrix.

### Source

Blanchard et al.

---

## knife_edge *Weight based knife-edge selectivity function*

---

### Description

A knife-edge selectivity function where weights greater or equal to `knife_edge_size` are selected.

### Usage

```
knife_edge(w, knife_edge_size, ...)
```

### Arguments

| | |
|---|---|
| w | The size of the individual. |
| knife_edge_size | The weight at which the knife-edge operates. |
| ... | Unused |

`lognormal_pred_kernel` *Lognormal predation kernel*

## Description

This is the most commonly-used predation kernel. The log of the predator/prey mass ratio is normally distributed.

## Usage

```
lognormal_pred_kernel(ppmr, beta, sigma)
```

## Arguments

| | |
|---|---|
| `ppmr` | A vector of predator/prey size ratios |
| `beta` | The preferred predator/prey size ratio |
| `sigma` | The width parameter of the log-normal kernel |

## Details

Writing the predator mass as $w$ and the prey mass as $w_p$, the feeding kernel is given as

$$\phi_i(w, w_p) = \exp\left[\frac{-(\ln(w/w_p/\beta_i))^2}{2\sigma_i^2}\right]$$

if $w/w_p$ is larger than 1 and zero otherwise. Here $\beta_i$ is the preferred predator-prey mass ratio and $\sigma_i$ determines the width of the kernel. These two parameters need to be given in the species parameter dataframe in the columns `beta` and `sigma`.

This function is called from `setPredKernel()` to set up the predation kernel slots in a MizerParams object.

## Value

A vector giving the value of the predation kernel at each of the predator/prey mass ratios in the `ppmr` argument.

`mizerEGrowth` *Get energy rate available for growth needed to project standard mizer model*

## Description

Calculates the energy rate $g_i(w)$ (grams/year) available by species and size for growth after metabolism, movement and reproduction have been accounted for. Used by `project()` for performing simulations. You would not usually call this function directly but instead use `getEGrowth()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerEGrowth(params, n, n_pp, n_other, t, e_repro, e, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `e_repro` | The energy available for reproduction as calculated by `getERepro()`. |
| `e` | The energy available for reproduction and growth as calculated by `getEReproAndGrowth()`. |
| `...` | Unused |

## Value

A two dimensional array (species x size) with the growth rates.

## Your own growth rate function

By default `getEGrowth()` calls `mizerEGrowth()`. However you can replace this with your own alternative growth rate function. If your function is called `"myEGrowth"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "EGrowth", "myEGrowth")
```

Your function will then be called instead of `mizerEGrowth()`, with the same arguments.

## See Also

Other mizer rate functions: `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

`mizerEncounter` *Get encounter rate needed to project standard mizer model*

## Description

Calculates the rate $E_i(w)$ at which a predator of species $i$ and weight $w$ encounters food (grams/year). You would not usually call this function directly but instead use `getEncounter()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerEncounter(params, n, n_pp, n_other, t, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The current time. Unused |
| `...` | Unused |

## Value

A named two dimensional array (predator species x predator size) with the encounter rates.

## Predation encounter

The encounter rate $E_i(w)$ at which a predator of species $i$ and weight $w$ encounters food has contributions from the encounter of fish prey and of resource. This is determined by summing over all prey species and the resource spectrum and then integrating over all prey sizes $w_p$, weighted by predation kernel $\phi(w, w_p)$:

$$E_i(w) = \gamma_i(w) \int \left( \theta_{ip} N_R(w_p) + \sum_j \theta_{ij} N_j(w_p) \right) \phi_i(w, w_p) w_p \, dw_p.$$

Here $N_j(w)$ is the abundance density of species $j$ and $N_R(w)$ is the abundance density of resource. The overall prefactor $\gamma_i(w)$ determines the predation power of the predator. It could be interpreted as a search volume and is set with the `setSearchVolume()` function. The predation kernel $\phi(w, w_p)$ is set with the `setPredKernel()` function. The species interaction matrix $\theta_{ij}$ is set with `setInteraction()` and the resource interaction vector $\theta_{ip}$ is taken from the `interaction_resource` column in `params@species_params`.

## Details

The encounter rate is multiplied by $1 - f_0$ to obtain the consumption rate, where $f_0$ is the feeding level calculated with `getFeedingLevel()`. This is used by the `project()` function for performing simulations.

The function returns values also for sizes outside the size-range of the species. These values should not be used, as they are meaningless.

If your model contains additional components that you added with `setComponent()` and for which you specified an `encounter_fun` function then the encounters of these components will be included in the returned value.

## Your own encounter function

By default `getEncounter()` calls `mizerEncounter()`. However you can replace this with your own alternative encounter function. If your function is called `"myEncounter"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "Encounter", "myEncounter")
```

Your function will then be called instead of `mizerEncounter()`, with the same arguments.

## See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

`mizerERepro` *Get energy rate available for reproduction needed to project standard mizer model*

---

## Description

Calculates the energy rate (grams/year) available for reproduction after growth and metabolism have been accounted for. You would not usually call this function directly but instead use `getERepro()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerERepro(params, n, n_pp, n_other, t, e, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |

| | |
|---|---|
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| e | A two dimensional array (species x size) holding the energy available for reproduction and growth as calculated by mizerEReproAndGrowth(). |
| ... | Unused |

## Value

A two dimensional array (species x size) holding

$$\psi_i(w) E_{r.i}(w)$$

where $E_{r.i}(w)$ is the rate at which energy becomes available for growth and reproduction, calculated with mizerEReproAndGrowth(), and $\psi_i(w)$ is the proportion of this energy that is used for reproduction. This proportion is taken from the params object and is set with setReproduction().

## Your own reproduction rate function

By default getERepro() calls mizerERepro(). However you can replace this with your own alternative reproduction rate function. If your function is called "myERepro" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "ERepro", "myERepro")
```

Your function will then be called instead of mizerERepro(), with the same arguments.

## See Also

Other mizer rate functions: mizerEGrowth(), mizerEReproAndGrowth(), mizerEncounter(), mizerFMortGear(), mizerFMort(), mizerFeedingLevel(), mizerMort(), mizerPredMort(), mizerPredRate(), mizerRDI(), mizerRates(), mizerResourceMort()

---

# mizerEReproAndGrowth *Get energy rate available for reproduction and growth needed to project standard mizer model*

---

## Description

Calculates the energy rate $E_{r.i}(w)$ (grams/year) available to an individual of species i and size w for reproduction and growth after metabolism and movement have been accounted for. You would not usually call this function directly but instead use getEReproAndGrowth(), which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerEReproAndGrowth(
  params,
  n,
  n_pp,
  n_other,
  t,
  encounter,
  feeding_level,
  ...
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| encounter | An array (species x size) with the encounter rate as calculated by getEncounter(). |
| feeding_level | An array (species x size) with the feeding level as calculated by getFeedingLevel(). |
| ... | Unused |

## Value

A two dimensional array (species x size) holding

$$E_{r.i}(w) = \max(0, \alpha_i\,(1 - \texttt{feeding\_level}_i(w))\,\texttt{encounter}_i(w) - \texttt{metab}_i(w)).$$

Due to the form of the feeding level, calculated by getFeedingLevel(), this can also be expressed as

$$E_{r.i}(w) = \max(0, \alpha_i\,\texttt{feeding\_level}_i(w)\,h_i(w) - \texttt{metab}_i(w))$$

where $h_i$ is the maximum intake rate, set with setMaxIntakeRate(). The assimilation rate $\alpha_i$ is taken from the species parameter data frame in params. The metabolic rate metab is taken from params and set with setMetabolicRate().

The return value can be negative, which means that the energy intake does not cover the cost of metabolism and movement.

## Your own energy rate function

By default getEReproAndGrowth() calls mizerEReproAndGrowth(). However you can replace this with your own alternative energy rate function. If your function is called "myEReproAndGrowth" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "EReproAndGrowth", "myEReproAndGrowth")
```

Your function will then be called instead of mizerEReproAndGrowth(), with the same arguments.

## See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

`mizerFeedingLevel` *Get feeding level needed to project standard mizer model*

---

## Description

You would not usually call this function directly but instead use `getFeedingLevel()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerFeedingLevel(params, n, n_pp, n_other, t, encounter, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The current time. Unused |
| `encounter` | A two dimensional array (predator species x predator size) with the encounter rate. |
| `...` | Unused |

## Value

A two dimensional array (predator species x predator size) with the feeding level.

## Feeding level

The feeding level $f_i(w)$ is the proportion of its maximum intake rate at which the predator is actually taking in fish. It is calculated from the encounter rate $E_i$ and the maximum intake rate $h_i(w)$ as

$$f_i(w) = \frac{E_i(w)}{E_i(w) + h_i(w)}.$$

The encounter rate $E_i$ is passed as an argument or calculated with `getEncounter()`. The maximum intake rate $h_i(w)$ is taken from the `params` object, and is set with `setMaxIntakeRate()`. As a consequence of the above expression for the feeding level, $1 - f_i(w)$ is the proportion of the food available to it that the predator actually consumes.

### Your own feeding level function

By default `getFeedingLevel()` calls `mizerFeedingLevel()`. However you can replace this with your own alternative feeding level function. If your function is called `"myFeedingLevel"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "FeedingLevel", "myFeedingLevel")
```

Your function will then be called instead of `mizerFeedingLevel()`, with the same arguments.

### See Also

The feeding level is used in `mizerEReproAndGrowth()` and in `mizerPredRate()`.

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

## mizerFMort — *Get the total fishing mortality rate from all fishing gears by time, species and size needed to project standard mizer model*

---

### Description

Calculates the total fishing mortality (in units 1/year) from all gears by species and size at each time step in the `effort` argument. The total fishing mortality is just the sum of the fishing mortalities imposed by each gear, $\mu_{f.i}(w) = \sum_g F_{g,i,w}$. You would not usually call this function directly but instead use `getFMort()`, which then calls this function unless an alternative function has been registered, see below.

### Usage

```
mizerFMort(params, effort, ...)
```

### Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `effort` | A vector with the effort for each fishing gear. |
| `...` | Unused |

### Value

An array (species x size) with the fishing mortality.

### Your own fishing mortality function

By default `getFMort()` calls `mizerFMort()`. However you can replace this with your own alternative fishing mortality function. If your function is called `"myFMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "FMort", "myFMort")
```

Your function will then be called instead of `mizerFMort()`, with the same arguments.

### Note

Here: fishing mortality = catchability x selectivity x effort.

### See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

## mizerFMortGear *Get the fishing mortality by time, gear, species and size needed to project standard mizer model*

---

### Description

Calculates the fishing mortality rate $F_{g,i,w}$ by gear, species and size at each time step in the `effort` argument (in units 1/year). This is a helper function for `mizerFMort()`.

### Usage

```
mizerFMortGear(params, effort)
```

### Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `effort` | A vector with the effort for each fishing gear. |

### Value

An three dimensional array (gear x species x size) with the fishing mortality

### Note

Here: fishing mortality = catchability x selectivity x effort.

## See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

`mizerMort` *Get total mortality rate needed to project standard mizer model*

---

## Description

Calculates the total mortality rate $\mu_i(w)$ (in units 1/year) on each species by size from predation mortality, background mortality and fishing mortality. You would not usually call this function directly but instead use `getMort()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerMort(params, n, n_pp, n_other, t, f_mort, pred_mort, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `f_mort` | A two dimensional array (species x size) with the fishing mortality |
| `pred_mort` | A two dimensional array (species x size) with the predation mortality |
| `...` | Unused |

## Details

If your model contains additional components that you added with `setComponent()` and for which you specified a `mort_fun` function then the mortality inflicted by these components will be included in the returned value.

## Value

A named two dimensional array (species x size) with the total mortality rates.

### Your own mortality function

By default `getMort()` calls `mizerMort()`. However you can replace this with your own alternative mortality function. If your function is called `"myMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "Mort", "myMort")
```

Your function will then be called instead of `mizerMort()`, with the same arguments.

### See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

## MizerParams *Alias for set_multispecies_model*

---

### Description

An alias provided for backward compatibility with mizer version <= 1.0

### Usage

```
MizerParams(
  species_params,
  interaction = matrix(1, nrow = nrow(species_params), ncol = nrow(species_params)),
  min_w_pp = 1e-10,
  min_w = 0.001,
  max_w = max(species_params$w_inf) * 1.1,
  no_w = 100,
  n = 2/3,
  q = 0.8,
  f0 = 0.6,
  kappa = 1e+11,
  lambda = 2 + q - n,
  r_pp = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| `species_params` | A data frame of species-specific parameter values. |
| `interaction` | Optional interaction matrix of the species (predator species x prey species). Entries should be numbers between 0 and 1. By default all entries are 1. See "Setting interactions" section below. |

| | |
|---|---|
| min_w_pp | The smallest size of the resource spectrum. By default this is set to the smallest value at which any of the consumers can feed. |
| min_w | Sets the size of the eggs of all species for which this is not given in the w_min column of the species_params dataframe. |
| max_w | The largest size of the consumer spectrum. By default this is set to the largest w_inf specified in the species_params data frame. |
| no_w | The number of size bins in the consumer spectrum. |
| n | The allometric growth exponent. This can be overruled for individual species by including a n column in the species_params. |
| q | Allometric exponent of search volume |
| f0 | Expected average feeding level. Used to set gamma, the coefficient in the search rate. Ignored if gamma is given explicitly. |
| kappa | Coefficient of the intrinsic resource carrying capacity |
| lambda | Scaling exponent of the intrinsic resource carrying capacity |
| r_pp | Coefficient of the intrinsic resource birth rate |
| ... | Unused |

## See Also

Other deprecated functions: set_community_model(), set_trait_model()

---

# MizerParams-class *A class to hold the parameters for a size based model.*

---

## Description

Although it is possible to build a MizerParams object by hand it is not recommended and several constructors are available. Dynamic simulations are performed using project() function on objects of this class. As a user you should never need to access the slots inside a MizerParams object directly.

## Slots

w The size grid for the fish part of the spectrum. An increasing vector of weights (in grams) running from the smallest egg size to the largest asymptotic size.

dw The widths (in grams) of the size bins

w_full The size grid for the full size range including the resource spectrum. An increasing vector of weights (in grams) running from the smallest resource size to the largest asymptotic size of fish. The last entries of the vector have to be equal to the content of the w slot.

dw_full The width of the size bins for the full spectrum. The last entries have to be equal to the content of the dw slot.

w_min_idx A vector holding the index of the weight of the egg size of each species

`maturity` An array (species x size) that holds the proportion of individuals of each species at size that are mature. This enters in the calculation of the spawning stock biomass with `getSSB()`. Set with `setReproduction()`.

`psi` An array (species x size) that holds the allocation to reproduction for each species at size, $\psi_i(w)$. Changed with `setReproduction()`.

`intake_max` An array (species x size) that holds the maximum intake for each species at size. Changed with `setMaxIntakeRate()`.

`search_vol` An array (species x size) that holds the search volume for each species at size. Changed with `setSearchVolume()`.

`metab` An array (species x size) that holds the metabolism for each species at size. Changed with `setMetabolicRate()`.

`mu_b` An array (species x size) that holds the external mortality rate $\mu_{b.i}(w)$. Changed with `setExtMort()`.

`pred_kernel` An array (species x predator size x prey size) that holds the predation coefficient of each predator at size on each prey size. If this is NA then the following two slots will be used. Changed with `setPredKernel()`.

`ft_pred_kernel_e` An array (species x log of predator/prey size ratio) that holds the Fourier transform of the feeding kernel in a form appropriate for evaluating the encounter rate integral. If this is NA then the `pred_kernel` will be used to calculate the available energy integral. Changed with `setPredKernel()`.

`ft_pred_kernel_p` An array (species x log of predator/prey size ratio) that holds the Fourier transform of the feeding kernel in a form appropriate for evaluating the predation mortality integral. If this is NA then the `pred_kernel` will be used to calculate the integral. Changed with `setPredKernel()`.

`rr_pp` A vector the same length as the w_full slot. The size specific growth rate of the resource spectrum. Changed with `setResource()`.

`cc_pp` A vector the same length as the w_full slot. The size specific carrying capacity of the resource spectrum. Changed with `setResource()`.

`resource_dynamics` Name of the function for projecting the resource abundance density by one timestep. The default is `resource_semichemostat()`. Changed with `setResource()`.

`other_dynamics` A named list of functions for projecting the values of other dynamical components of the ecosystem that may be modelled by a mizer extensions you have installed. The names of the list entries are the names of those components.

`other_encounter` A named list of functions for calculating the contribution to the encounter rate from each other dynamical component.

`other_mort` A named list of functions for calculating the contribution to the mortality rate from each other dynamical components.

`other_params` A list containing the parameters needed by any mizer extensions you may have installed to model other dynamical components of the ecosystem.

`rates_funcs` A named list with the names of the functions that should be used to calculate the rates needed by `project()`. By default this will be set to the names of the built-in rate functions.

`sc` The community abundance of the scaling community

`species_params` A data.frame to hold the species specific parameters. See `newMultispeciesParams()` for details.

`gear_params` Data frame with parameters for gear selectivity. See `setFishing()` for details.

`interaction` The species specific interaction matrix, $\theta_{ij}$. Changed with `setInteraction()`.

`selectivity` An array (gear x species x w) that holds the selectivity of each gear for species and size, $S_{g,i,w}$. Changed with `setFishing()`.

`catchability` An array (gear x species) that holds the catchability of each species by each gear, $Q_{g,i}$. Changed with `setFishing()`.

`initial_effort` A vector containing the initial fishing effort for each gear. Changed with `setFishing()`.

`initial_n` An array (species x size) that holds the initial abundance of each species at each weight.

`initial_n_pp` A vector the same length as the w_full slot that describes the initial resource abundance at each weight.

`initial_n_other` A list with the initial abundances of all other ecosystem components. Has length zero if there are no other components.

`resource_params` List with parameters for resource. See `setResource()`.

`A` Abundance multipliers.

`linecolour` A named vector of colour values, named by species. Used to give consistent colours in plots.

`linetype` A named vector of linetypes, named by species. Used to give consistent line types in plots.

`ft_mask` An array (species x w_full) with zeros for weights larger than the asymptotic weight of each species. Used to efficiently minimize wrap-around errors in Fourier transform calculations.

The MizerParams class is fairly complex with a large number of slots, many of which are multidimensional arrays. The dimensions of these arrays is strictly enforced so that `MizerParams` objects are consistent in terms of number of species and number of size classes.

The `MizerParams` class does not hold any dynamic information, e.g. abundances or harvest effort through time. These are held in MizerSim objects.

## See Also

`project()` `MizerSim()` `emptyParams()` `newMultispeciesParams()` `newCommunityParams()` `newTraitParams()`

---

`mizerPredMort` *Get total predation mortality rate needed to project standard mizer model*

---

## Description

Calculates the total predation mortality rate $\mu_{p,i}(w_p)$ (in units of 1/year) on each prey species by prey size:

$$\mu_{p.i}(w_p) = \sum_j \texttt{pred\_rate}_j(w_p)\, \theta_{ji}.$$

You would not usually call this function directly but instead use `getPredMort()`, which then calls this function unless an alternative function has been registered, see below.

### Usage

```
mizerPredMort(params, n, n_pp, n_other, t, pred_rate, ...)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| pred_rate | A two dimensional array (predator species x predator size) with the feeding level. |
| ... | Unused |

### Value

A two dimensional array (prey species x prey size) with the predation mortality

### Your own predation mortality function

By default `getPredMort()` calls `mizerPredMort()`. However you can replace this with your own alternative predation mortality function. If your function is called `"myPredMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "PredMort", "myPredMort")
```

Your function will then be called instead of `mizerPredMort()`, with the same arguments.

### See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`, `mizerResourceMort()`

---

## mizerPredRate *Get predation rate needed to project standard mizer model*

---

### Description

Calculates the potential rate (in units 1/year) at which a prey individual of a given size $w$ is killed by predators from species $j$. In formulas

$$\texttt{pred\_rate}_j(w_p) = \int \phi_j(w, w_p)(1 - f_j(w))\gamma_j(w)N_j(w)\, dw.$$

This potential rate is used in the function `mizerPredMort()` to calculate the realised predation mortality rate on the prey individual. You would not usually call this function directly but instead use `getPredRate()`, which then calls this function unless an alternative function has been registered, see below.

### Usage

```
mizerPredRate(params, n, n_pp, n_other, t, feeding_level, ...)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| feeding_level | An array (species x size) with the feeding level as calculated by getFeedingLevel(). |
| ... | Unused |

### Value

A named two dimensional array (predator species x prey size) with the predation rate, where the prey size runs over fish community plus resource spectrum.

### Your own predation rate function

By default getPredRate() calls mizerPredRate(). However you can replace this with your own alternative predation rate function. If your function is called "myPredRate" then you register it in a MizerParams object params with

```
params <- setRateFunction(params, "PredRate", "myPredRate")
```

Your function will then be called instead of mizerPredRate(), with the same arguments.

### See Also

Other mizer rate functions: mizerEGrowth(), mizerEReproAndGrowth(), mizerERepro(), mizerEncounter(), mizerFMortGear(), mizerFMort(), mizerFeedingLevel(), mizerMort(), mizerPredMort(), mizerRDI(), mizerRates(), mizerResourceMort()

---

## mizerRates *Get all rates needed to project standard mizer model*

---

### Description

Calls other rate functions in sequence and collects the results in a list.

### Usage

```
mizerRates(params, n, n_pp, n_other, t = 0, effort, rates_fns, ...)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size). |
| n_pp | A vector of the resource abundance by size |
| n_other | A list of abundances for other dynamical components of the ecosystem |
| t | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| effort | The effort for each fishing gear |
| rates_fns | Named list of the functions to call to calculate the rates. Note that this list holds the functions themselves, not their names. |
| ... | Unused |

## Details

By default this function returns a list with the following components:

- encounter from mizerEncounter()
- feeding_level from mizerFeedingLevel()
- pred_rate from mizerPredRate()
- pred_mort from mizerPredMort()
- f_mort from mizerFMort()
- mort from mizerMort()
- resource_mort from mizerResourceMort()
- e from mizerEReproAndGrowth()
- e_repro from mizerERepro()
- e_growth from mizerEGrowth()
- rdi from mizerRDI()
- rdd from BevertonHoltRDD()

However you can replace any of these rate functions by your own rate function if you wish, see setRateFunction() for details.

## See Also

Other mizer rate functions: mizerEGrowth(), mizerEReproAndGrowth(), mizerERepro(), mizerEncounter(), mizerFMortGear(), mizerFMort(), mizerFeedingLevel(), mizerMort(), mizerPredMort(), mizerPredRate(), mizerRDI(), mizerResourceMort()

`mizerRDI` *Get density-independent rate of reproduction needed to project standard mizer model*

## Description

Calculates the density-independent rate of total egg production $R_{p.i}$ (units 1/year) before density dependence, by species. This is obtained by taking the total rate at which energy is invested in reproduction, multiplying by the reproductive efficiency `erepro` and dividing by the egg size `w_min`, and by a factor of two to account for the two sexes. You would not usually call this function directly but instead use `getRDI()`, which then calls this function unless an alternative function has been registered, see below.

## Usage

```
mizerRDI(params, n, n_pp, n_other, t, e_growth, mort, e_repro, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `e_growth` | An array (species x size) with the energy available for growth as calculated by `getEGrowth()`. Unused. |
| `mort` | An array (species x size) with the mortality rate as calculated by `getMort()`. Unused. |
| `e_repro` | An array (species x size) with the energy available for reproduction as calculated by `getERepro()`. |
| `...` | Unused |

## Details

Used by `getRDD()` to calculate the actual, density dependent rate. See `setReproduction()` for more details.

## Value

A numeric vector with the rate of egg production for each species.

### Your own reproduction function

By default `getRDI()` calls `mizerRDI()`. However you can replace this with your own alternative reproduction function. If your function is called `"myRDI"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "RDI", "myRDI")
```

Your function will then be called instead of `mizerRDI()`, with the same arguments.

### See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRates()`, `mizerResourceMort()`

---

## mizerResourceMort *Get predation mortality rate for resource needed to project standard mizer model*

---

### Description

Calculates the predation mortality rate $\mu_p(w)$ on the resource spectrum by resource size (in units 1/year). You would not usually call this function directly but instead use `getResourceMort()`, which then calls this function unless an alternative function has been registered, see below.

### Usage

```
mizerResourceMort(params, n, n_pp, n_other, t, pred_rate, ...)
```

### Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size). |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list of abundances for other dynamical components of the ecosystem |
| `t` | The time for which to do the calculation (Not used by standard mizer rate functions but useful for extensions with time-dependent parameters.) |
| `pred_rate` | A two dimensional array (predator species x prey size) with the predation rate, where the prey size runs over fish community plus resource spectrum. |
| `...` | Unused |

### Value

A vector of mortality rate by resource size.

### Your own resource mortality function

By default `getResourceMort()` calls `mizerResourceMort()`. However you can replace this with your own alternative resource mortality function. If your function is called `"myResourceMort"` then you register it in a MizerParams object `params` with

```
params <- setRateFunction(params, "ResourceMort", "myResourceMort")
```

Your function will then be called instead of `mizerResourceMort()`, with the same arguments.

### See Also

Other mizer rate functions: `mizerEGrowth()`, `mizerEReproAndGrowth()`, `mizerERepro()`, `mizerEncounter()`, `mizerFMortGear()`, `mizerFMort()`, `mizerFeedingLevel()`, `mizerMort()`, `mizerPredMort()`, `mizerPredRate()`, `mizerRDI()`, `mizerRates()`

---

## `MizerSim` *Constructor for the* `MizerSim` *class*

---

### Description

A constructor for the `MizerSim` class. This is used by `project()` to create `MizerSim` objects of the right dimensions. It is not necessary for users to use this constructor.

### Usage

```
MizerSim(params, t_dimnames = NA, t_max = 100, t_save = 1)
```

### Arguments

| | |
|---|---|
| `params` | a MizerParams object |
| `t_dimnames` | Numeric vector that is used for the time dimensions of the slots. Default = NA. |
| `t_max` | The maximum time step of the simulation. Only used if t_dimnames = NA. Default value = 100. |
| `t_save` | How often should the results of the simulation be stored. Only used if t_dimnames = NA. Default value = 1. |

### Value

An object of type MizerSim

`MizerSim-class` *A class to hold the results of a simulation*

## Description

A class that holds the results of projecting a MizerParams object through time using `project()`.

## Details

A new `MizerSim` object can be created with the `MizerSim()` constructor, but you will never have to do that because the object is created automatically by `project()` when needed.

As a user you should never have to access the slots of a MizerSim object directly. Instead there are a range of functions to extract the information. `N()` and `NResource()` return arrays with the saved abundances of the species and the resource population at size respectively. `getEffort()` returns the fishing effort of each gear through time. `getTimes()` returns the vector of times at which simulation results were stored and `idxFinalT()` returns the index with which to access specifically the value at the final time in the arrays returned by the other functions. `getParams()` returns the `MizerParams` object that was passed to `project()`. There are also several summary_functions and plotting_functions available to explore the contents of a `MizerSim` object.

The arrays all have named dimensions. The names of the `time` dimension denote the time in years. The names of the `w` dimension are weights in grams rounded to three significant figures. The names of the `sp` dimension are the same as the species name in the order specified in the species_params data frame. The names of the `gear` dimension are the names of the gears, in the same order as specified when setting up the `MizerParams` object.

Extensions of mizer can use the `n_other` slot to store the abundances of other ecosystem components and these extensions should provide their own functions for accessing that information.

The `MizerSim` class has changed since previous versions of mizer. To use a `MizerSim` object created by a previous version, you need to upgrade it with `upgradeSim()`.

## Slots

`params` An object of type MizerParams.

`n` Three-dimensional array (time x species x size) that stores the projected community number densities.

`n_pp` An array (time x size) that stores the projected resource number densities.

`n_other` A list array (time x component) that stores the projected values for other ecosystem components.

`effort` An array (time x gear) that stores the fishing effort by time and gear.

## N *Time series of size spectra*

### Description

Fetch the simulation results for the size spectra over time.

### Usage

```
N(sim)

NResource(sim)
```

### Arguments

| | |
|---|---|
| sim | A MizerSim object |

### Value

For `N()`: A three-dimensional array (time x species x size) with the number density of consumers

For `NResource()`: An array (time x size) with the number density of resource

## newCommunityParams *Set up parameters for a community-type model*

### Description

This functions creates a `MizerParams` object describing a community-type model.

### Usage

```
newCommunityParams(
  max_w = 1e+06,
  min_w = 0.001,
  no_w = 100,
  min_w_pp = 1e-10,
  z0 = 0.1,
  alpha = 0.2,
  f0 = 0.7,
  h = 10,
  gamma = NA,
  beta = 100,
  sigma = 2,
  n = 2/3,
  kappa = 1000,
```

```
  lambda = 2.05,
  r_pp = 10,
  knife_edge_size = 1000,
  reproduction
)
```

## Arguments

| | |
|---|---|
| max_w | The maximum size of the community. The w_inf of the species used to represent the community is set to this value. |
| min_w | The minimum size of the community. |
| no_w | The number of size bins in the consumer spectrum. |
| min_w_pp | The smallest size of the resource spectrum. By default this is set to the smallest value at which any of the consumers can feed. |
| z0 | The background mortality of the community. |
| alpha | The assimilation efficiency of the community. |
| f0 | The average feeding level of individuals who feed on a power-law spectrum. This value is used to calculate the search rate parameter gamma. |
| h | The coefficient of the maximum food intake rate. |
| gamma | Volumetric search rate. Estimated using h, f0 and kappa if not supplied. |
| beta | The preferred predator prey mass ratio. |
| sigma | The width of the prey preference. |
| n | The allometric growth exponent. Used as allometric exponent for the maximum intake rate of the community as well as the intrinsic growth rate of the resource. |
| kappa | Coefficient of the intrinsic resource carrying capacity |
| lambda | Scaling exponent of the intrinsic resource carrying capacity |
| r_pp | Coefficient of the intrinsic resource birth rate |
| knife_edge_size | The size at the edge of the knife-edge-selectivity function. |
| reproduction | The constant reproduction in the smallest size class of the community spectrum. By default this is set so that the community spectrum is continuous with the resource spectrum. |

## Details

A community model has several features that distinguish it from a multi-species model:

- Species identities of individuals are ignored. All are aggregated into a single community.
- The resource spectrum only extends to the start of the community spectrum.
- Reproductive rate is constant, independent of the energy invested in reproduction, which is set to 0.
- Standard metabolism is turned off (the parameter ks is set to 0). Consequently, the growth rate is now determined solely by the assimilated food

The function has many arguments, all of which have default values.

Fishing selectivity is modelled as a knife-edge function with one parameter, `knife_edge_size`, which determines the size at which species are selected.

The resulting `MizerParams` object can be projected forward using `project()` like any other `MizerParams` object. When projecting the community model it may be necessary to keep a small time step size `dt` of around 0.1 to avoid any instabilities with the solver. You can check for these numerical instabilities by plotting the biomass or abundance through time after the projection.

## Value

An object of type `MizerParams`

## References

K. H. Andersen,J. E. Beyer and P. Lundberg, 2009, Trophic and individual efficiencies of size-structured communities, Proceedings of the Royal Society, 276, 109-114

## See Also

Other functions for setting up models: `newMultispeciesParams()`, `newTraitParams()`

## Examples

```
## Not run:
params <- newCommunityParams(f0=0.7, z0=0.2, reproduction=3e7)
sim <- project(params, effort = 0, t_max = 100, dt=0.1)
plotBiomass(sim)
plotSpectra(sim)

## End(Not run)
```

---

# newMultispeciesParams *Set up parameters for a general multispecies model*

---

## Description

Sets up a multi-species size spectrum model by filling all slots in the MizerParams object based on user-provided or default parameters. It does this by creating an empty MizerParams object with `emptyParams()` and then filling the slots by passing its arguments to `setParams()`. There is a long list of arguments, but almost all of them have sensible default values. All arguments are described in more details in the sections below the list.

## Usage

```
newMultispeciesParams(
  species_params,
  interaction = NULL,
  no_w = 100,
  min_w = 0.001,
  max_w = NA,
  min_w_pp = NA,
  pred_kernel = NULL,
  search_vol = NULL,
  intake_max = NULL,
  metab = NULL,
  p = 0.7,
  z0 = NULL,
  z0pre = 0.6,
  z0exp = n - 1,
  maturity = NULL,
  repro_prop = NULL,
  RDD = "BevertonHoltRDD",
  resource_rate = NULL,
  resource_capacity = NULL,
  n = 2/3,
  r_pp = 10,
  kappa = 1e+11,
  lambda = 2.05,
  w_pp_cutoff = 10,
  resource_dynamics = "resource_semichemostat",
  gear_params = data.frame(),
  selectivity = NULL,
  catchability = NULL,
  initial_effort = NULL
)
```

## Arguments

| | |
|---|---|
| species_params | A data frame of species-specific parameter values. |
| interaction | Optional interaction matrix of the species (predator species x prey species). Entries should be numbers between 0 and 1. By default all entries are 1. See "Setting interactions" section below. |
| no_w | The number of size bins in the consumer spectrum. |
| min_w | Sets the size of the eggs of all species for which this is not given in the w_min column of the species_params dataframe. |
| max_w | The largest size of the consumer spectrum. By default this is set to the largest w_inf specified in the species_params data frame. |
| min_w_pp | The smallest size of the resource spectrum. By default this is set to the smallest value at which any of the consumers can feed. |

pred_kernel Optional. An array (species x predator size x prey size) that holds the predation coefficient of each predator at size on each prey size. If not supplied, a default is set as described in section "Setting predation kernel".

search_vol Optional. An array (species x size) holding the search volume for each species at size. If not supplied, a default is set as described in the section "Setting search volume".

intake_max Optional. An array (species x size) holding the maximum intake rate for each species at size. If not supplied, a default is set as described in the section "Setting maximum intake rate".

metab Optional. An array (species x size) holding the metabolic rate for each species at size. If not supplied, a default is set as described in the section "Setting metabolic rate".

p The allometric metabolic exponent. This is only used if metab is not given explicitly and if the exponent is not specified in a p column in the species_params.

z0 Optional. An array (species x size) holding the external mortality rate.

z0pre If z0, the mortality from other sources, is not a column in the species data frame, it is calculated as z0pre * w_inf ^ z0exp. Default value is 0.6.

z0exp If z0, the mortality from other sources, is not a column in the species data frame, it is calculated as z0pre * w_inf ^ z0exp. Default value is n-1.

maturity Optional. An array (species x size) that holds the proportion of individuals of each species at size that are mature. If not supplied, a default is set as described in the section "Setting reproduction".

repro_prop Optional. An array (species x size) that holds the proportion of consumed energy that a mature individual allocates to reproduction for each species at size. If not supplied, a default is set as described in the section "Setting reproduction".

RDD The name of the function calculating the density-dependent reproduction rate from the density-independent rate. Defaults to "BevertonHoltRDD()".

resource_rate Optional. Vector of resource intrinsic birth rates

resource_capacity
Optional. Vector of resource intrinsic carrying capacity

n The allometric growth exponent. This can be overruled for individual species by including a n column in the species_params.

r_pp Coefficient of the intrinsic resource birth rate

kappa Coefficient of the intrinsic resource carrying capacity

lambda Scaling exponent of the intrinsic resource carrying capacity

w_pp_cutoff The upper cut off size of the resource spectrum. Default is 10 g.

resource_dynamics
Function that determines resource dynamics by calculating the resource spectrum at the next time step from the current state.

gear_params A data frame with gear-specific parameter values.

selectivity An array (gear x species x size) that holds the selectivity of each gear for species and size, $S_{g,i,w}$.

`catchability` An array (gear x species) that holds the catchability of each species by each gear, $Q_{g,i}$.

`initial_effort` Optional. A number or a named numeric vector specifying the fishing effort. If a number, the same effort is used for all gears. If a vector, must be named by gear.

## Value

An object of type MizerParams

## Species parameters

The only essential argument is a data frame that contains the species parameters. The data frame is arranged species by parameter, so each column of the parameter data frame is a parameter and each row has the values of the parameters for one of the species in the model.

There are two essential columns that must be included in the species parameter data.frame and that do not have default values: the `species` column that should hold strings with the names of the species and the `w_inf` column with the asymptotic sizes of the species.

The species_params dataframe also needs to contain the parameters needed by any predation kernel function or size selectivity function. This will be mentioned in the appropriate sections below.

For all other species parameters, mizer will calculate default values if they are not included in the species parameter data frame. They will be automatically added when the `MizerParams` object is created. For these parameters you can also specify values for only some species and leave the other entries as NA and the missing values will be set to the defaults.

All the parameters will be mentioned in the following sections.

## Changes to species params

The `species_params` slot of the returned MizerParams object may differ slightly from the data frame supplied as argument to this function in the following ways:

- Default values are set for `w_min`,`w_inf`,`alpha`,`gear`,`interaction_resource`.
- The egg sizes in `w_min` are rounded down to lie on a grid point.

Note that the other characteristic sizes of the species, like `w_mat` and `w_inf`, are not modified to lie on grid points.

## Size grid

A size grid is created so that the log-sizes are equally spaced. The spacing is chosen so that there will be `no_w` fish size bins, with the smallest starting at `min_w` and the largest starting at `max_w`. For `w_full` additional size bins are added below `min_w`, with the same log size. The number of extra bins is such that `min_w_pp` comes to lie within the smallest bin.

### Units in mizer

Mizer uses grams to measure weight, centimetres to measure lengths, and years to measure time.

Mizer is agnostic about whether abundances are given as

1. numbers per area,
2. numbers per volume or
3. total numbers for the entire study area.

You should make the choice most convenient for your application and then stick with it. If you make choice 1 or 2 you will also have to choose a unit for area or volume. Your choice will then determine the units for some of the parameters. This will be mentioned when the parameters are discussed in the sections below.

You choice will also affect the units of the quantities you may want to calculate with the model. For example, the yield will be in grams/year/m^2 in case 1 if you choose m^2 as your measure of area, in grams/year/m^3 in case 2 if you choose m^3 as your unit of volume, or simply grams/year in case 3. The same comment applies for other measures, like total biomass, which will be grams/area in case 1, grams/volume in case 2 or simply grams in case 3. When mizer puts units on axes, for example in `plotBiomass`, it will simply put grams, as appropriate for case 3.

You can convert between these choices. For example, if you use case 1, you need to multiply with the area of the ecosystem to get the total quantity. If you work with case 2, you need to multiply by both area and the thickness of the productive layer. In that respect, case 2 is a bit cumbersome.

### Setting interactions

The interaction matrix $\theta_{ij}$ 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 encountered food and predation mortality (see on the website the section on predator-prey encounter rate and on predation mortality).

It is used when calculating the food encounter rate in `getEncounter()` and the predation mortality rate in `getPredMort()`. Its entries are dimensionless numbers.The values are between 0 (species do not overlap and therefore do not interact with 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.

This function checks that the supplied interaction matrix is valid and then stores it in the `interaction` slot of the params object before returning that object.

The order of the columns and rows of the `interaction` argument should be the same as the order in the species params data frame in the `params` object. If you supply a named array then the function will check the order and warn if it is different. One way of creating your own interaction matrix is to enter the data using a spreadsheet program and saving it as a .csv file. The data can be read into R using the command `read.csv()`.

The interaction of the species with the resource are set via a column `interaction_resource` in the `species_params` data frame. Again the entries have to be numbers between 0 and 1. By default this column is set to all 1s.

### Setting predation kernel

#### Kernel dependent on predator to prey size ratio

If the `pred_kernel` argument is not supplied, then this function sets a predation kernel that depends only on the ratio of predator mass to prey mass, not on the two masses independently. The shape of that kernel is then determined by the `pred_kernel_type` column in species_params.

The default pred_kernel_type is "lognormal". This will call the function `lognormal_pred_kernel()` to calculate the predation kernel. An alternative pred_kernel type is "box", implemented by the function `box_pred_kernel()`, and "power_law", implemented by the function `power_law_pred_kernel()`. These functions require certain species parameters in the species_params data frame. For the log-normal kernel these are `beta` and `sigma`, for the box kernel they are `ppmr_min` and `ppmr_max`. They are explained in the help pages for the kernel functions. Except for `beta` and `sigma`, no defaults are set for these parameters. If they are missing from the species_params data frame then mizer will issue an error message.

You can use any other string as the type. If for example you choose "my" then you need to define a function `my_pred_kernel` that you can model on the existing functions like `lognormal_pred_kernel()`.

When using a kernel that depends on the predator/prey size ratio only, mizer does not need to store the entire three dimensional array in the MizerParams object. Such an array can be very big when there is a large number of size bins. Instead, mizer only needs to store two two-dimensional arrays that hold Fourier transforms of the feeding kernel function that allow the encounter rate and the predation rate to be calculated very efficiently. However, if you need the full three-dimensional array you can calculate it with the `getPredKernel()` function.

#### Kernel dependent on both predator and prey size

If you want to work with a feeding kernel that depends on predator mass and prey mass independently, you can specify the full feeding kernel as a three-dimensional array (predator species x predator size x prey size).

You should use this option only if a kernel dependent only on the predator/prey mass ratio is not appropriate. Using a kernel dependent on predator/prey mass ratio only allows mizer to use fast Fourier transform methods to significantly reduce the running time of simulations.

The order of the predator species in `pred_kernel` should be the same as the order in the species params dataframe in the `params` object. If you supply a named array then the function will check the order and warn if it is different.

### Setting search volume

The search volume $\gamma_i(w)$ of an individual of species $i$ and weight $w$ multiplies the predation kernel when calculating the encounter rate in `getEncounter()` and the predation rate in `getPredRate()`.

The name "search volume" is a bit misleading, because $\gamma_i(w)$ does not have units of volume. It is simply a parameter that determines the rate of predation. Its units depend on your choice, see section "Units in mizer". If you have chose to work with total abundances, then it is a rate with units 1/year. If you have chosen to work with abundances per m^2 then it has units of m^2/year. If you have chosen to work with abundances per m^3 then it has units of m^3/year.

If the `search_vol` argument is not supplied, then the search volume is set to

$$\gamma_i(w) = \gamma_i w_i^q.$$

The values of $\gamma_i$ (the search volume at 1g) and $q_i$ (the allometric exponent of the search volume) are taken from the gamma and q columns in the species parameter dataframe. If the gamma column is not supplied in the species parameter dataframe, a default is calculated by the get_gamma_default() function. Note that only for predators of size $w = 1$ gram is the value of the species parameter $\gamma_i$ the same as the value of the search volume $\gamma_i(w)$.

### Setting maximum intake rate

The maximum intake rate $h_i(w)$ of an individual of species $i$ and weight $w$ determines the feeding level, calculated with getFeedingLevel(). It is measured in grams/year.

If the intake_max argument is not supplied, then the maximum intake rate is set to

$$h_i(w) = h_i w_i^n.$$

The values of $h_i$ (the maximum intake rate of an individual of size 1 gram) and $n_i$ (the allometric exponent for the intake rate) are taken from the h and n columns in the species parameter dataframe. If the h column is not supplied in the species parameter dataframe, it is calculated by the get_h_default() function, using f0 and the k_vb column, if they are supplied.

If $h_i$ is set to Inf, fish will consume all encountered food.

### Setting metabolic rate

The metabolic rate is subtracted from the energy income rate to calculate the rate at which energy is available for growth and reproduction, see getEReproAndGrowth(). It is measured in grams/year.

If the metab argument is not supplied, then for each species the metabolic rate $k(w)$ for an individual of size $w$ is set to

$$k(w) = ksw^p + kw,$$

where $ksw^p$ represents the rate of standard metabolism and $kw$ is the rate at which energy is expended on activity and movement. The values of $ks$, $p$ and $k$ are taken from the ks, p and k columns in the species parameter dataframe. If any of these parameters are not supplied, the defaults are $k = 0$, $p = n$ and

$$ks = f_c h \alpha w_{mat}^{n-p},$$

where $f_c$ is the critical feeding level taken from the fc column in the species parameter data frame. If the critical feeding level is not specified, a default of $f_c = 0.2$ is used.

### Setting external mortality rate

The external mortality is all the mortality that is not due to fishing or predation by predators included in the model. The external mortality could be due to predation by predators that are not explicitly included in the model (e.g. mammals or seabirds) or due to other causes like illness. It is a rate with units 1/year.

The z0 argument allows you to specify an external mortality rate that depends on species and body size. You can see an example of this in the Examples section of the help page for setExtMort().

If the z0 argument is not supplied, then the external mortality is assumed to depend only on the species, not on the size of the individual: $\mu_{b.i}(w) = z_{0.i}$. The value of the constant $z_0$ for each species is taken from the z0 column of the species_params data frame, if that column exists. Otherwise it is calculated as

$$z_{0.i} = \texttt{z0pre}_i \, w_{inf}^{\texttt{z0exp}}.$$

### Setting reproduction

**Investment:** For each species and at each size, the proportion $\psi$ of the available energy that is invested into reproduction is the product of two factors: the proportion `maturity` of individuals that are mature and the proportion `repro_prop` of the energy available to a mature individual that is invested into reproduction.

If the `maturity` argument is not supplied, then it is set to a sigmoidal maturity ogive that changes from 0 to 1 at around the maturity size:

$$\texttt{maturity}(w) = \left[1 + \left(\frac{w}{w_{mat}}\right)^{-U}\right]^{-1}.$$

(To avoid clutter, we are not showing the species index in the equations.) The maturity weights are taken from the `w_mat` column of the species_params data frame. Any missing maturity weights are set to 1/4 of the asymptotic weight in the `w_inf` column. The exponent $U$ determines the steepness of the maturity ogive. By default it is chosen as $U = 10$, however this can be overridden by including a column `w_mat25` in the species parameter dataframe that specifies the weight at which 25% of individuals are mature, which sets $U = \log(3)/\log(w_{mat}/w_{25})$.

The sigmoidal function given above would strictly reach 1 only asymptotically. Mizer instead sets the function equal to 1 already at the species' maximum size, taken from the compulsory `w_inf` column in the `species_params` data frame.

If the `repro_prop` argument is not supplied, it is set to the allometric form

$$\texttt{repro\_prop}(w) = \left(\frac{w}{w_{inf}}\right)^{m-n}.$$

Here $n$ is the scaling exponent of the energy income rate. Hence the exponent $m$ determines the scaling of the investment into reproduction for mature individuals. By default it is chosen to be $m = 1$ so that the rate at which energy is invested into reproduction scales linearly with the size. This default can be overridden by including a column `m` in the species parameter dataframe. The asymptotic sizes are taken from the compulsory `w_inf` column in the species_params data frame.

So finally we have

$$\psi(w) = \texttt{maturity}(w)\texttt{repro\_prop}(w)$$

**Efficiency:** The reproductive efficiency, i.e., the proportion of energy allocated to reproduction that results in egg biomass, is set through the `erepro` column in the species_params data frame. If that is not provided, the default is set to 1 (which you will want to override). The offspring biomass divided by the egg biomass gives the rate of egg production, returned by `getRDI()`.

**Density dependence:** The stock-recruitment relationship is an emergent phenomenon in mizer, with several sources of density dependence. Firstly, the amount of energy invested into reproduction depends on the energy income of the spawners, which is density-dependent due to competition for prey. Secondly, the proportion of larvae that grow up to recruitment size depends on the larval mortality, which depends on the density of predators, and on larval growth rate, which depends on density of prey.

Finally, to encode all the density dependence in the stock-recruitment relationship that is not already included in the other two sources of density dependence, mizer puts the the density-independent rate of egg production through a density-dependence function. The result is returned by `getRDD()`. The name of the density-dependence function is specified by the `RDD` argument.

The default is the Beverton-Holt function `BevertonHoltRDD()`, which requires an `R_max` column in the species_params data frame giving the maximum egg production rate. If this column does not exist, it is initialised to `Inf`, leading to no density-dependence. Other functions provided by mizer are `RickerRDD()` and `SheperdRDD()` and you can easily use these as models for writing your own functions.

## Setting fishing

### Gears

In `mizer`, fishing mortality is imposed on species by fishing gears. The total per-capita fishing mortality (1/year) is obtained by summing over the mortality from all gears,

$$\mu_{f.i}(w) = \sum_g F_{g,i}(w),$$

where the fishing mortality $F_{g,i}(w)$ imposed by gear $g$ on species $i$ at size $w$ is calculated as:

$$F_{g,i}(w) = S_{g,i}(w) Q_{g,i} E_g,$$

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.

### Selectivity

The selectivity at size of each gear for each species is saved as a three dimensional array (gear x species x size). Each entry has a range between 0 (that gear is not selecting that species at that size) to 1 (that gear is selecting all individuals of that species of that size). This three dimensional array can be specified explicitly via the `selectivity` argument, but usually mizer calculates it from the `gear_params` slot of the MizerParams object.

To allow the calculation of the `selectivity` array, the `gear_params` slot must be a data frame with one row for each gear-species combination. So if for example a gear can select three species, then that gear contributes three rows to the `gear_params` data frame, one for each species it can select. The data frame must have columns `gear`, holding the name of the gear, `species`, holding the name of the species, and `sel_func`, holding the name of the function that calculates the selectivity curve. Some selectivity functions are included in the package: `knife_edge()`, `sigmoid_length()`, `double_sigmoid_length()`, and `sigmoid_weight()`. 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.

Each selectivity function may have parameters. Values for these parameters must be included as columns in the gear parameters data.frame. The names of the columns must exactly match the names of the corresponding arguments of the selectivity function. For example, the default selectivity function is `knife_edge()` that a has sudden change of selectivity from 0 to 1 at a certain size. In its help page you can see that the `knife_edge()` function has arguments `w` and `knife_edge_size`. 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 gear parameters data.frame. So for the `knife_edge()` function there should be a `knife_edge_size` column. Because `knife_edge()` is the default selectivity function, the `knife_edge_size` argument has a default value = `w_mat`.

In case each species is only selected by one gear, the columns of the `gear_params` data frame can alternatively be provided as columns of the `species_params` data frame, if this is more convenient

for the user to set up. Mizer will then copy these columns over to create the `gear_params` data frame when it creates the MizerParams object. However changing these columns in the species parameter data frame later will not update the `gear_params` data frame.

**Catchability**

Catchability is used as an additional factor 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.

Catchability is stored as a two dimensional array (gear x species). This can either be provided explicitly via the `catchability` argument, or the information can be provided via a `catchability` column in the `gear_params` data frame.

In the case where each species is selected by only a single gear, the `catchability` column can also be provided in the `species_params` data frame. Mizer will then copy this over to the `gear_params` data frame when the MizerParams object is created.

**Effort**

The initial fishing effort is stored in the `MizerParams` object. If it is not supplied, it is set to zero. The initial effort can be overruled when the simulation is run with `project()`, where it is also possible to specify an effort that varies through time.

## Setting resource dynamics

By default, mizer uses a semichemostat model to describe the resource dynamics in each size class independently. This semichemostat dynamics is implemented by the function `resource_semichemostat()`. You can change the resource dynamics by writing your own function, modelled on `resource_semichemostat()`, and then passing the name of your function in the `resource_dynamics` argument.

The `resource_rate` argument is a vector specifying the intrinsic resource growth rate for each size class. If it is not supplied, then the intrinsic growth rate $r(w)$ at size $w$ is set to

$$r(w) = r_{pp}\, w^{n-1}.$$

The values of $r_{pp}$ and $n$ are taken from the `r_pp` and `n` arguments.

The `resource_capacity` argument is a vector specifying the intrinsic resource carrying capacity for each size class. If it is not supplied, then the intrinsic carrying capacity $c(w)$ at size $w$ is set to

$$c(w) = \kappa\, w^{-\lambda}$$

for all $w$ less than `w_pp_cutoff` and zero for larger sizes. The values of $\kappa$ and $\lambda$ are taken from the `kappa` and `lambda` arguments.

## See Also

Other functions for setting up models: `newCommunityParams()`, `newTraitParams()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter)

## End(Not run)
```

`newTraitParams` *Set up parameters for a trait-based model*

## Description

This functions creates a `MizerParams` object describing a trait-based model. This is a simplification of the general size-based model used in `mizer` in which the species-specific parameters are the same for all species, except for the asymptotic size, which is considered the most important trait characterizing a species. Other parameters are related to the asymptotic size. For example, the size at maturity is given by `w_inf * eta`, where `eta` is the same for all species. For the trait-based model the number of species is not important. For applications of the trait-based model see Andersen & Pedersen (2010). See the `mizer` website for more details and examples of the trait-based model.

## Usage

```
newTraitParams(
  no_sp = 11,
  min_w_inf = 10,
  max_w_inf = 10^4,
  min_w = 10^(-3),
  max_w = max_w_inf,
  eta = 10^(-0.6),
  min_w_mat = min_w_inf * eta,
  no_w = log10(max_w_inf/min_w) * 20 + 1,
  min_w_pp = 1e-10,
  w_pp_cutoff = min_w_mat,
  n = 2/3,
  p = n,
  lambda = 2.05,
  r_pp = 0.1,
  kappa = 0.005,
  alpha = 0.4,
  h = 40,
  beta = 100,
  sigma = 1.3,
  f0 = 0.6,
  fc = 0.25,
  ks = NA,
  gamma = NA,
  ext_mort_prop = 0,
  R_factor = 4,
  gear_names = "knife_edge_gear",
  knife_edge_size = 1000,
  egg_size_scaling = FALSE,
  resource_scaling = FALSE,
  perfect_scaling = FALSE
)
```

## Arguments

| | |
|---|---|
| no_sp | The number of species in the model. |
| min_w_inf | The asymptotic size of the smallest species in the community. This will be rounded to lie on a grid point. |
| max_w_inf | The asymptotic size of the largest species in the community. This will be rounded to lie on a grid point. |
| min_w | The size of the the egg of the smallest species. This also defines the start of the community size spectrum. |
| max_w | The largest size in the model. By default this is set to the largest asymptotic size max_w_inf. Setting it to something larger only makes sense if you plan to add larger species to the model later. |
| eta | Ratio between maturity size and asymptotic size of a species. Ignored if min_w_mat is supplied. Default is 10^(-0.6), approximately 1/4. |
| min_w_mat | The maturity size of the smallest species. Default value is eta * min_w_inf. This will be rounded to lie on a grid point. |
| no_w | The number of size bins in the community spectrum. These bins will be equally spaced on a logarithmic scale. Default value is such that there are 50 bins for each factor of 10 in weight. |
| min_w_pp | The smallest size of the resource spectrum. By default this is set to the smallest value at which any of the consumers can feed. |
| w_pp_cutoff | The largest size of the resource spectrum. Default value is min_w_inf unless perfect_scaling = TRUE when it is Inf. |
| n | Scaling exponent of the maximum intake rate. |
| p | Scaling exponent of the standard metabolic rate. By default this is equal to the exponent n. |
| lambda | Exponent of the abundance power law. |
| r_pp | Growth rate parameter for the resource spectrum. |
| kappa | Coefficient in abundance power law. |
| alpha | The assimilation efficiency of the community. |
| h | Maximum food intake rate. |
| beta | Preferred predator prey mass ratio. |
| sigma | Width of prey size preference. |
| f0 | Expected average feeding level. Used to set gamma, the coefficient in the search rate. Ignored if gamma is given explicitly. |
| fc | Critical feeding level. Used to determine ks if it is not given explicitly. |
| ks | Standard metabolism coefficient. If not provided, default will be calculated from critical feeding level argument fc. |
| gamma | Volumetric search rate. If not provided, default is determined by get_gamma_default() using the value of f0. |
| ext_mort_prop | The proportion of the total mortality that comes from external mortality, i.e., from sources not explicitly modelled. A number in the interval [0, 1). |

R_factor The factor such that R_max = R_factor * R, where R_max is the maximum reproduction rate allowed and R is the steady-state reproduction rate. Thus the larger R_factor the less the impact of the density-dependence.

gear_names The names of the fishing gears for each species. A character vector, the same length as the number of species.

knife_edge_size
The minimum size at which the gear or gears select fish. A single value for each gear or a vector with one value for each gear.

egg_size_scaling
If TRUE, the egg size is a constant fraction of the maximum size of each species. This fraction is min_w / min_w_inf. If FALSE, all species have the egg size w_min.

resource_scaling
If TRUE, the carrying capacity for larger resource is reduced to compensate for the fact that fish eggs and larvae are present in the same size range.

perfect_scaling
If TRUE then parameters are set so that the community abundance, growth before reproduction and death are perfect power laws. In particular all other scaling corrections are turned on.

## Details

The function has many arguments, all of which have default values. Of particular interest to the user are the number of species in the model and the minimum and maximum asymptotic sizes.

The characteristic weights of the smallest species are defined by min_w (egg size), min_w_mat (maturity size) and min_w_inf (asymptotic size). The asymptotic sizes of the no_sp species are logarithmically evenly spaced, ranging from min_w_inf to max_w_inf. Similarly the maturity sizes of the species are logarithmically evenly spaced, so that the ratio eta between maturity size and asymptotic size is the same for all species. If egg_size_scaling = TRUE then also the ratio between asymptotic size and egg size is the same for all species. Otherwise all species have the same egg size.

In addition to setting up the parameters, this function also sets up an initial condition that is close to steady state.

Although the trait based model's steady state is often stable without imposing additional density-dependence, the function can set a Beverton-Holt type density-dependence that imposes a maximum for the reproduction rate that is a multiple of the reproduction rate at steady state. That multiple is set by the argument R_factor.

The search rate coefficient gamma is calculated using the expected feeding level, f0.

The option of including fishing is given, but the steady state may lose its natural stability if too much fishing is included. In such a case the user may wish to include stabilising effects (like R_factor) to ensure the steady state is stable. Fishing selectivity is modelled as a knife-edge function with one parameter, knife_edge_size, which is the size at which species are selected. Each species can either be fished by the same gear (knife_edge_size has a length of 1) or by a different gear (the length of knife_edge_size has the same length as the number of species and the order of selectivity size is that of the asymptotic size).

The resulting MizerParams object can be projected forward using project() like any other MizerParams object. When projecting the model it may be necessary to reduce dt below 0.1 to avoid any instabilities with the solver. You can check this by plotting the biomass or abundance through time after the projection.

## Value

An object of type MizerParams

## See Also

Other functions for setting up models: newCommunityParams(), newMultispeciesParams()

## Examples

```
## Not run:
params <- newTraitParams()
sim <- project(params, t_max = 5, effort = 0)
plotSpectra(sim)

## End(Not run)
```

---

# noRDD *Give density-independent reproduction rate*

---

## Description

Simply returns its rdi argument.

## Usage

```
noRDD(rdi, ...)
```

## Arguments

| | |
|---|---|
| rdi | Vector of density-independent reproduction rates $R_p$ for all species. |
| ... | Not used. |

## Value

Vector of density-dependent reproduction rates.

## See Also

Other functions calculating density-dependent reproduction rate: BevertonHoltRDD(), RickerRDD(), SheperdRDD(), constantRDD()

## NOther *Time series of other components*

### Description

Fetch the simulation results for other components over time.

### Usage

```
NOther(sim)
```

### Arguments

| | |
|---|---|
| sim | A MizerSim object |

### Value

A list array (time x component) that stores the projected values for other ecosystem components.

## NS_params *Example MizerParams object for the North Sea example*

### Description

A MizerParams object created from the `NS_species_params_gears` species parameters and the `inter` interaction matrix together with an initial condition corresponding to the steady state obtained from fishing with an effort `effort = c(Industrial = 0, Pelagic = 1, Beam = 0.5, Otter = 0.5)`.

### Format

A MizerParams object

### Source

Blanchard et al.

### Examples

```
## Not run:
sim = project(NS_params, effort = c(Industrial = 0, Pelagic = 1,
                                    Beam = 0.5, Otter = 0.5))
plot(sim)

## End(Not run)
```

## NS_species_params *Example species parameter set based on the North Sea*

### Description

This data set is based on species in the North Sea (Blanchard et al.). It is a data.frame that contains all the necessary information to be used by the `MizerParams()` constructor. As there is no gear column, each species is assumed to be fished by a separate gear.

### Format

A data frame with 12 rows and 7 columns. Each row is a species.

**species** Name of the species

**w_inf** The von Bertalanffy W_infinity parameter

**w_mat** Size at maturity

**beta** Size preference ratio

**sigma** Width of the size-preference

**R_max** Maximum reproduction rate

**k_vb** The von Bertalanffy k parameter

### Source

Blanchard et al.

### Examples

```
## Not run:
params <- MizerParams(NS_species_params)
sim = project(params)
plot(sim)

## End(Not run)
```

## NS_species_params_gears *Example species parameter set based on the North Sea with different gears*

### Description

This data set is based on species in the North Sea (Blanchard et al.). It is similar to the data set `NS_species_params` except that this one has an additional column specifying the fishing gear that operates on each species.

### Format

A data frame with 12 rows and 8 columns. Each row is a species.

**species** Name of the species

**w_inf** The von Bertalanffy W_infinity parameter

**w_mat** Size at maturity

**beta** Size preference ratio

**sigma** Width of the size-preference

**R_max** Maximum reproduction rate

**k_vb** The von Bertalanffy k parameter

**gear** Name of the fishing gear

### Source

Blanchard et al.

### Examples

```
## Not run:
params <- MizerParams(NS_species_params_gears)
sim = project(params, effort = c(Industrial = 0, Pelagic = 1,
                                 Beam = 0.5, Otter = 0.5))
plot(sim)

## End(Not run)
```

---

## plot,MizerSim,missing-method

*Summary plot for* `MizerSim` *objects*

---

### Description

After running a projection, produces 5 plots in the same window: feeding level, abundance spectra, predation mortality and fishing mortality of each species by size; and biomass of each species through time. This method just uses the other plotting functions and puts them all in one window.

Produces 3 plots in the same window: abundance spectra, feeding level and predation mortality of each species through time. This method just uses the other plotting functions and puts them all in one window.

### Usage

```
## S4 method for signature 'MizerSim,missing'
plot(x, y, ...)

## S4 method for signature 'MizerParams,missing'
plot(x, y, ...)
```

### Arguments

| | |
|---|---|
| x | An object of class MizerSim |
| y | Not used |
| ... | For additional arguments see the documentation for plotBiomass(), plotFeedingLevel(),plotSpectra and plotFMort(). |

### Value

A viewport object

A viewport object

### See Also

plotting_functions

plotting_functions

Other plotting functions: plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

Other plotting functions: plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

### Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plot(sim)
plot(sim, time_range = 10:20) # change time period for size-based plots
plot(sim, min_w = 10, max_w = 1000) # change size range for biomass plot


params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
plot(params)
plot(params, min_w = 10, max_w = 1000) # change size range for biomass plot
```

---

## plotBiomass *Plot the biomass of species through time*

---

### Description

After running a projection, the biomass of each species can be plotted against time. The biomass is calculated within user defined size limits (min_w, max_w, min_l, max_l, see getBiomass()).

## Usage

```
plotBiomass(
  sim,
  species,
  start_time,
  end_time,
  y_ticks = 6,
  ylim = c(NA, NA),
  total = FALSE,
  background = TRUE,
  highlight = NULL,
  ...
)

plotlyBiomass(
  sim,
  species,
  start_time,
  end_time,
  y_ticks = 6,
  ylim = c(NA, NA),
  total = FALSE,
  background = TRUE,
  highlight = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| sim | An object of class MizerSim |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| start_time | The first time to be plotted. Default is the beginning of the time series. |
| end_time | The last time to be plotted. Default is the end of the time series. |
| y_ticks | The approximate number of ticks desired on the y axis |
| ylim | A numeric vector of length two providing lower and upper limits for the y axis. Use NA to refer to the existing minimum or maximum. Any values below 1e-20 are always cut off. |
| total | A boolean value that determines whether the total biomass from all species is plotted as well. Default is FALSE. |
| background | A boolean value that determines whether background species are included. Ignored if the model does not contain background species. Default is TRUE. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Arguments passed on to get_size_range_array<br>min_w Smallest weight in size range. Defaults to smallest weight in the model. |

max_w Largest weight in size range. Defaults to largest weight in the model.

min_l Smallest length in size range. If supplied, this takes precedence over min_w.

max_l Largest length in size range. If supplied, this takes precedence over max_w.

## Value

A plot

## See Also

plotting_functions, getBiomass()

Other plotting functions: plot,MizerSim,missing-method, plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

## Examples

```
# Set up example MizerParams and MizerSim objects
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort = 1, t_max = 20, t_save = 0.2, progress_bar = FALSE)

plotBiomass(sim)
plotBiomass(sim, species = c("Cod", "Haddock"), total = TRUE)
plotBiomass(sim, min_w = 10, max_w = 1000)
plotBiomass(sim, start_time = 10, end_time = 15)
plotBiomass(sim, y_ticks = 3)
```

---

# plotDiet *Plot diet*

---

## Description

Plot diet

## Usage

```
plotDiet(object, species)
```

## Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | The name of the species whose diet should be plotted |

## Value

A plot

## See Also

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

---

plotFeedingLevel *Plot the feeding level of species by size*

---

## Description

After running a projection, plot the feeding level of each species by size. The feeding level is averaged over the specified time range (a single value for the time range can be used).

## Usage

```
plotFeedingLevel(
  object,
  species = NULL,
  time_range,
  highlight = NULL,
  all.sizes = FALSE,
  include_critical = FALSE,
  ...
)

plotlyFeedingLevel(
  object,
  species = NULL,
  time_range,
  highlight = NULL,
  include_critical,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| time_range | The time range (either a vector of values, a vector of min and max time, or a single value) to average the abundances over. Default is the final time step. Ignored when called with a MizerParams object. |
| highlight | Name or vector of names of the species to be highlighted. |
| all.sizes | If TRUE, then feeding level is plotted also for sizes outside a species' size range. Default FALSE. |
| include_critical | If TRUE, then the critical feeding level is also plotted. Default FALSE. |
| ... | Other arguments (currently unused) |

## Details

When called with a MizerSim object, the feeding level is averaged over the specified time range (a single value for the time range can be used to plot a single time step). When called with a MizerParams object the initial feeding level is plotted.

If `include_critical = TRUE` then the critical feeding level (the feeding level at which the intake just covers the metabolic cost) is also plotted, with a thinner line. This line should always stay below the line of the actual feeding level, because the species would stop growing at any point where the feeding level drops to the critical feeding level.

## Value

A plot

## See Also

plotting_functions, `getFeedingLevel()`

Other plotting functions: `plot,MizerSim,missing-method`, `plotBiomass()`, `plotDiet()`, `plotFMort()`, `plotGrowthCurves()`, `plotPredMort()`, `plotSpectra()`, `plotYieldGear()`, `plotYield()`, `plotting_functions`

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotFeedingLevel(sim)
plotFeedingLevel(sim, time_range = 10:20, species = c("Cod", "Herring"),
                 include_critical = TRUE)
```

---

`plotFMort` *Plot total fishing mortality of each species by size*

---

## Description

After running a projection, plot the total fishing mortality of each species by size. The total fishing mortality is averaged over the specified time range (a single value for the time range can be used to plot a single time step).

## Usage

```
plotFMort(object, species = NULL, time_range, highlight = NULL, ...)

plotlyFMort(object, species = NULL, time_range, highlight = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| time_range | The time range (either a vector of values, a vector of min and max time, or a single value) to average the abundances over. Default is the final time step. Ignored when called with a MizerParams object. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

### Value

A plot

### See Also

plotting_functions, getFMort()

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

### Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotFMort(sim)
plotFMort(sim, highlight = c("Cod", "Haddock"))
```

---

## plotGrowthCurves *Plot growth curves giving weight as a function of age*

---

### Description

When the growth curve for only a single species is plotted, horizontal lines are included that indicate the maturity size and the maximum size for that species. If furthermore the species parameters contain the variables a and b for length to weight conversion and the von Bertalanffy parameter k_vb (and optionally t0), then the von Bertalanffy growth curve is superimposed in black.

### Usage

```
plotGrowthCurves(
  object,
  species,
  max_age = 20,
  percentage = FALSE,
```

```
  highlight = NULL
)

plotlyGrowthCurves(
  object,
  species,
  max_age = 20,
  percentage = FALSE,
  highlight = NULL
)
```

## Arguments

| | |
|---|---|
| object | MizerSim or MizerParams object. If given a MizerSim object, uses the growth rates at the final time of a simulation to calculate the size at age. If given a MizerParams object, uses the initial growth rates instead. |
| species | Name or vector of names of the species to be included. By default all species are included. |
| max_age | The age up to which to run the growth curve. Default is 20. |
| percentage | Boolean value. If TRUE, the size is given as a percentage of the maximal size. |
| highlight | Name or vector of names of the species to be highlighted. |

## Value

A plot

## See Also

plotting_functions

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotGrowthCurves(sim, percentage = TRUE)
plotGrowthCurves(sim, species = "Cod", max_age = 24)
```

plotM2 *Alias for plotPredMort*

## Description

An alias provided for backward compatibility with mizer version <= 1.0

## Usage

```
plotM2(object, species = NULL, time_range, highlight = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| time_range | The time range (either a vector of values, a vector of min and max time, or a single value) to average the abundances over. Default is the final time step. Ignored when called with a MizerParams object. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

## Value

A plot

## See Also

plotting_functions, getPredMort()

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotPredMort(sim)
plotPredMort(sim, time_range = 10:20)
```

plotPredMort *Plot predation mortality rate of each species against size*

## Description

After running a projection, plot the predation mortality rate of each species by size. The mortality rate is averaged over the specified time range (a single value for the time range can be used to plot a single time step).

## Usage

```
plotPredMort(object, species = NULL, time_range, highlight = NULL, ...)

plotlyPredMort(object, species = NULL, time_range, highlight = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| time_range | The time range (either a vector of values, a vector of min and max time, or a single value) to average the abundances over. Default is the final time step. Ignored when called with a MizerParams object. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

## Value

A plot

## See Also

plotting_functions, getPredMort()

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotSpectra(), plotYieldGear(), plotYield(), plotting_functions

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotPredMort(sim)
plotPredMort(sim, time_range = 10:20)
```

plotSpectra *Plot the abundance spectra*

## Description

Plots the number density multiplied by a power of the weight, with the power specified by the power argument.

## Usage

```
plotSpectra(
  object,
  species = NULL,
  time_range,
  wlim = c(NA, NA),
  ylim = c(NA, NA),
  power = 1,
  biomass = TRUE,
  total = FALSE,
  resource = TRUE,
  background = TRUE,
  highlight = NULL,
  ...
)

plotlySpectra(
  object,
  species = NULL,
  time_range,
  wlim = c(NA, NA),
  ylim = c(NA, NA),
  power = 1,
  biomass = TRUE,
  total = FALSE,
  resource = TRUE,
  background = TRUE,
  highlight = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class MizerSim or MizerParams. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |

| | |
|---|---|
| time_range | The time range (either a vector of values, a vector of min and max time, or a single value) to average the abundances over. Default is the final time step. Ignored when called with a MizerParams object. |
| wlim | A numeric vector of length two providing lower and upper limits for the w axis. Use NA to refer to the existing minimum or maximum. |
| ylim | A numeric vector of length two providing lower and upper limits for the y axis. Use NA to refer to the existing minimum or maximum. Any values below 1e-20 are always cut off. |
| power | The abundance is plotted as the number density times the weight raised to power. The default power = 1 gives the biomass density, whereas power = 2 gives the biomass density with respect to logarithmic size bins. |
| biomass | Obsolete. Only used if power argument is missing. Then biomass = TRUE is equivalent to power=1 and biomass = FALSE is equivalent to power=0 |
| total | A boolean value that determines whether the total over all species in the system is plotted as well. Default is FALSE |
| resource | A boolean value that determines whether resource is included. Default is TRUE. |
| background | A boolean value that determines whether background species are included. Ignored if the model does not contain background species. Default is TRUE. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

## Details

When called with a MizerSim object, the abundance is averaged over the specified time range (a single value for the time range can be used to plot a single time step). When called with a MizerParams object the initial abundance is plotted.

## Value

A plot

## See Also

plotting_functions

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotYieldGear(), plotYield(), plotting_functions

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)
plotSpectra(sim)
plotSpectra(sim, wlim = c(1e-6, NA))
plotSpectra(sim, time_range = 10:20)
plotSpectra(sim, time_range = 10:20, power = 0)
```

```
plotSpectra(sim, species = c("Cod", "Herring"), power = 1)
```

---

`plotting_functions` *Description of the plotting functions*

---

## Description

Mizer provides a range of plotting functions for visualising the results of running a simulation, stored in a MizerSim object, or the initial state stored in a MizerParams object. Every plotting function exists in two versions, `plotSomething` and `plotlySomething`. The plotly version is more interactive but not suitable for inclusion in documents.

## Details

This table shows the available plotting functions.

| Plot | Description |
|---|---|
| `plotBiomass()` | Plots the total biomass of each species through time. A time range to be plotted can be specified. The |
| `plotSpectra()` | Plots the abundance (biomass or numbers) spectra of each species and the background community. It |
| `plotFeedingLevel()` | Plots the feeding level of each species against size. |
| `plotPredMort()` | 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 species by gear against time. |
| `plotDiet()` | Plots the diet composition at size for a given predator species. |
| `plotGrowthCurves()` | Plots the size as a function of age. |
| `plot()` | Produces 5 plots (`plotFeedingLevel()`, `plotBiomass()`, `plotPredMort()`, `plotFMort()` and `plo` |

These functions use the ggplot2 package and return the plot as a ggplot object. This means that you can manipulate the plot further after its creation using the ggplot grammar of graphics. The corresponding function names with `plot` replaced by `plotly` produce interactive plots with the help of the plotly package.

While most plot functions take their data from a MizerSim object, some of those that make plots representing data at a single time can also take their data from the initial values in a MizerParams object.

Where plots show results for species, the line colour and line type for each species are specified by the `linecolour` and `linetype` slots in the MizerParams object. These were either taken from a default palette hard-coded into `emptyParams()` or they were specified by the user in the species parameters dataframe used to set up the MizerParams object. The `linecolour` and `linetype` slots hold named vectors, named by the species. They can be overwritten by the user at any time.

Most plots allow the user to select to show only a subset of species, specified as a vector in the `species` argument to the plot function.

The ordering of the species in the legend is the same as the ordering in the species parameter data frame.

## See Also

summary_functions, indicator_functions

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotYield()

## Examples

```
# Set up example MizerParams and MizerSim objects
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 2, progress_bar = FALSE)

# Some example plots
plotFeedingLevel(sim)

# Plotting only a subset of species
plotFeedingLevel(sim, species = c("Cod", "Herring"))

# Specifying new colours and linetypes for some species
sim@params@linetype["Cod"] <- "solid"
sim@params@linecolour["Cod"] <- "red"
plotFeedingLevel(sim, species = c("Cod", "Herring"))

# Manipulating the plot
library(ggplot2)
p <- plotFeedingLevel(sim)
p <- p + geom_hline(aes(yintercept = 0.7))
p <- p + theme_bw()
p
```

---

# plotYield *Plot the total yield of species through time*

---

## Description

After running a projection, the total yield of each species across all fishing gears can be plotted against time. The yield is obtained with getYield().

## Usage

```
plotYield(sim, sim2, species, total = FALSE, log = TRUE, highlight = NULL, ...)

plotlyYield(
  sim,
  sim2,
  species,
```

```
  total = FALSE,
  log = TRUE,
  highlight = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| sim | An object of class MizerSim |
| sim2 | An optional second object of class MizerSim. If this is provided its yields will be shown on the same plot in bolder lines. |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| total | A boolean value that determines whether the total over all species in the system is plotted as well. Default is FALSE |
| log | Boolean whether yield should be plotted on a logarithmic axis. Defaults to true. |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

## Value

A plot

## See Also

plotting_functions, getYield()

Other plotting functions: plot,MizerSim,missing-method, plotBiomass(), plotDiet(), plotFMort(), plotFeedingLevel(), plotGrowthCurves(), plotPredMort(), plotSpectra(), plotYieldGear(), plotting_functions

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 0.2, progress_bar = FALSE)
plotYield(sim)
plotYield(sim, species = c("Cod", "Herring"), total = TRUE)

# Comparing with yield from twice the effort
sim2 <- project(params, effort=2, t_max=20, t_save = 0.2, progress_bar = FALSE)
plotYield(sim, sim2, species = c("Cod", "Herring"), log = FALSE)
```

plotYieldGear *Plot the total yield of each species by gear through time*

## Description

After running a projection, the total yield of each species by fishing gear can be plotted against time.

## Usage

```
plotYieldGear(sim, species, total = FALSE, highlight = NULL, ...)

plotlyYieldGear(sim, species, total = FALSE, highlight = NULL, ...)
```

## Arguments

| | |
|---|---|
| sim | An object of class MizerSim |
| species | Name or vector of names of the species to be plotted. By default all species are plotted. |
| total | A boolean value that determines whether the total over all species in the system is plotted as well. Default is FALSE |
| highlight | Name or vector of names of the species to be highlighted. |
| ... | Other arguments (currently unused) |

## Details

This plot is pretty easy to do by hand. It just gets the biomass using the `getYieldGear()` method and plots using the ggplot2 package. You can then fiddle about with colours and linetypes etc. Just look at the source code for details.

## Value

A plot

## See Also

plotting_functions, `getYieldGear()`

Other plotting functions: `plot,MizerSim,missing-method`, `plotBiomass()`, `plotDiet()`, `plotFMort()`, `plotFeedingLevel()`, `plotGrowthCurves()`, `plotPredMort()`, `plotSpectra()`, `plotYield()`, `plotting_functions`

## Examples

```
params <- suppressMessages(newMultispeciesParams(NS_species_params_gears, inter))
sim <- project(params, effort=1, t_max=20, t_save = 0.2, progress_bar = FALSE)
plotYieldGear(sim)
plotYieldGear(sim, species = c("Cod", "Herring"), total = TRUE)
```

`power_law_pred_kernel` *Power-law predation kernel*

## Description

This predation kernel is a power-law, with sigmoidal cut-offs at large and small predator/prey mass ratios.

## Usage

```
power_law_pred_kernel(
  ppmr,
  kernel_exp,
  kernel_l_l,
  kernel_u_l,
  kernel_l_r,
  kernel_u_r
)
```

## Arguments

| | |
|---|---|
| `ppmr` | A vector of predator/prey size ratios at which to evaluate the predation kernel. |
| `kernel_exp` | The exponent of the power law |
| `kernel_l_l` | The location of the left, rising sigmoid |
| `kernel_u_l` | The shape of the left, rising sigmoid |
| `kernel_l_r` | The location of the right, falling sigmoid |
| `kernel_u_r` | The shape of the right, falling sigmoid |

## Details

The return value is calculated as

```
ppmr^kernel_exp / (1 + (exp(kernel_l_l) / ppmr)^kernel_u_l) / (1 + (ppmr / exp(kernel_l_r))^kernel_u_r)
```

The parameters need to be given as columns in the species parameter dataframe.

## Value

A vector giving the value of the predation kernel at each of the predator/prey mass ratios in the `ppmr` argument.

`project` *Project size spectrum forward in time*

## Description

Runs the size spectrum model simulation. The function returns an object of type MizerSim that can then be explored with a range of summary_functions, indicator_functions and plotting_functions.

## Usage

```
project(
  object,
  effort,
  t_max = 100,
  dt = 0.1,
  t_save = 1,
  t_start = 0,
  initial_n,
  initial_n_pp,
  append = TRUE,
  progress_bar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | Either a MizerParams object or a MizerSim object (which contains a `MizerParams` object). |
| `effort` | The effort of each fishing gear through time. See notes below. |
| `t_max` | The number of years the projection runs for. The default value is 100. However, this argument is ignored if an array is used for the `effort` argument. See notes below. |
| `dt` | Time step of the solver. The default value is 0.1. |
| `t_save` | The frequency with which the output is stored. The default value is 1. Must be an integer multiple of dt. |
| `t_start` | The the year of the start of the simulation. The simulation will cover the period from `t_start` to `t_start + t_max`. Defaults to 0. Ignored if an array is used for the `effort` argument or a `MizerSim` for the `object` argument. |
| `initial_n` | Deprecated. The initial abundances of species. Instead of using this argument you should set `initialN(params)` to the desired value. |
| `initial_n_pp` | Deprecated. The initial abundances of resource. Instead of using this argument you should set `initialNResource(params)` to the desired value. |
| `append` | A boolean that determines whether the new simulation results are appended to the previous ones. Only relevant if `object` is a `MizerSim` object. Default = TRUE. |

| | |
|---|---|
| progress_bar | Either a boolean value to determine whether a progress bar should be shown in the console, or a shiny Progress object to implement a progress bar in a shiny app. |
| ... | Other arguments will be passed to rate functions. |

## Value

An object of class MizerSim.

## Note

The effort argument specifies the level of fishing effort during the simulation. If it is not supplied, the initial effort stored in the params object is used. The effort can be specified in three different ways:

- A single numeric value. This specifies the effort of all fishing gears which is constant through time (i.e. all the gears have the same constant effort).
- A numerical vector which has the same length as the number of fishing gears. The vector must be named and the names must correspond to the gear names in the MizerParams object. The values in the vector specify the constant fishing effort of each of the fishing gears, i.e. the effort is constant through time but each gear may have a different fishing effort.
- A numerical array with dimensions time x gear. This specifies the fishing effort of each gear at each time step. The first dimension, time, must be named numerically and increasing. The second dimension of the array must be named and the names must correspond to the gear names in the MizerParams argument. The value for the effort for a particular time is used during the interval from that time to the next time in the array.

If effort is specified as an array then the smallest time in the array is used as the initial time for the simulation. Otherwise the initial time is set to the final time of the previous simulation if object is a MizerSim object or to t_start otherwise. Also, if the effort is an array then the t_max argument is ignored and the maximum simulation time is the largest time of the effort array.

If the object argument is of class MizerSim then the initial values for the simulation are taken from the final values in the MizerSim object and the corresponding arguments to this function will be ignored.

## Examples

```
## Not run:
# Data set with different fishing gears
params <- newMultispeciesParams(NS_species_params_gears, inter)
# With constant fishing effort for all gears for 20 time steps
sim <- project(params, t_max = 20, effort = 0.5)
# With constant fishing effort which is different for each gear
effort <- c(Industrial = 0, Pelagic = 1, Beam = 0.5, Otter = 0.5)
sim <- project(params, t_max = 20, effort = effort)
# With fishing effort that varies through time for each gear
gear_names <- c("Industrial","Pelagic","Beam","Otter")
times <- seq(from = 1, to = 10, by = 1)
effort_array <- array(NA, dim = c(length(times), length(gear_names)),
    dimnames = list(time = times, gear = gear_names))
```

```
effort_array[,"Industrial"] <- 0.5
effort_array[,"Pelagic"] <- seq(from = 1, to = 2, length = length(times))
effort_array[,"Beam"] <- seq(from = 1, to = 0, length = length(times))
effort_array[,"Otter"] <- seq(from = 1, to = 0.5, length = length(times))
sim <- project(params, effort = effort_array)

## End(Not run)
```

---

`project_simple` *Project abundances by a given number of time steps into the future*

---

## Description

This is an internal function used by the user-facing `project()` function. It is of potential interest only to mizer extension authors.

## Usage

```
project_simple(
  params,
  n,
  n_pp,
  n_other,
  t,
  dt,
  steps,
  effort,
  resource_dynamics_fn,
  other_dynamics_fns,
  rates_fns,
  ...
)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object. |
| `n` | An array (species x size) with the number density at start of simulation. |
| `n_pp` | A vector (size) with the resource number density at start of simulation. |
| `n_other` | A named list with the abundances of other components at start of simulation. |
| `t` | Time at the start of the simulation. |
| `dt` | Size of time step. |
| `steps` | The number of time steps by which to project. |
| `effort` | The fishing effort to be used throughout the simulation. This must be a vector or list with one named entry per fishing gear. |
| `resource_dynamics_fn` | The function for the resource dynamics. See Details. |

other_dynamics_fns
: List with the functions for the dynamics of the other components. See Details.

rates_fns
: List with the functions for calculating the rates. See Details.

...
: Other arguments that are passed on to the rate functions.

## Details

The function does not check its arguments because it is mean to be as fast as possible to allow it to be used in a loop. For example it is called in `project()` once for every saved value. The function also does not save its intermediate results but only returns the result at time `t + dt * steps`. During this time it uses the constant fishing effort `effort`.

The functional arguments can be calculated from slots in the `params` object with

```
resource_dynamics_fn <- get(params@resource_dynamics)
other_dynamics_fns <- lapply(params@other_dynamics, get)
rates_fns <- lapply(params@rates_funcs, get)
```

The reason the function does not do that itself is to shave 20 microseconds of its running time, which pays when the function is called hundreds of times in a row.

This function is also used in `steady()`. In between calls to `project_simple()` the `steady()` function checks whether the values are still changing significantly, so that it can stop when a steady state has been approached. Mizer extension packages might have a similar need to run a simulation repeatedly for short periods to run some other code in between. Because this code may want to use the values of the rates at the final time step, these too are included in the returned list.

## Value

List with the final values of `n`, `n_pp` and `n_other`, `rates`.

---

# resource_constant *Keep resource abundance constant*

---

## Description

This function can be used instead of the standard `resource_semichemostat()` in order to keep the Resource spectrum constant over time.

## Usage

```
resource_constant(params, n, n_pp, n_other, rates, t, dt, ...)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size) |
| n_pp | A vector of the resource abundance by size |
| n_other | A list with the abundances of other components |
| rates | A list of rates as returned by `mizerRates()` |
| t | The current time |
| dt | Time step |
| ... | Unused |

### Value

Vector containing resource spectrum at next timestep

### Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter,
                                resource_dynamics = "resource_constant")

## End(Not run)
```

---

## resource_encounter *Dummy function used during testing only*

---

### Description

Dummy function used during testing only

### Usage

```
resource_encounter(params, n, n_pp, n_other, ...)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| n | A matrix of species abundances (species x size) |
| n_pp | A vector of the resource abundance by size |
| n_other | A list with the abundances of other components |
| ... | Unused |

`resource_semichemostat` *Project resource using semichemostat model*

## Description

Project resource using semichemostat model

## Usage

```
resource_semichemostat(params, n, n_pp, n_other, rates, t, dt, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n` | A matrix of species abundances (species x size) |
| `n_pp` | A vector of the resource abundance by size |
| `n_other` | A list with the abundances of other components |
| `rates` | A list of rates as returned by `mizerRates()` |
| `t` | The current time |
| `dt` | Time step |
| `...` | Unused |

## Value

Vector containing resource spectrum at next timestep

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears, inter,
                                resource_dynamics = "resource_semichemostat")

## End(Not run)
```

## retune_erepro *Retune reproduction efficiency to maintain initial egg abundances*

### Description

Sets the reproductive efficiency for all species so that the rate of egg production exactly compensates for the loss from the first size class due to growth and mortality. Turns off the external density dependence in the reproduction rate by setting the RDD function to noRDD()

### Usage

```
retune_erepro(params, species = species_params(params)$species)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| species | A vector of the names of the species to be affected or a boolean vector indicating for each species whether it is to be affected (TRUE) or not. By default all species are affected |

### Value

A MizerParams object

## RickerRDD *Ricker function to calculate density-dependent reproduction rate*

### Description

Takes the density-independent rates $R_p$ of egg production and returns reduced, density-dependent rates $R$ given as

$$R = R_p \exp -bR_p$$

### Usage

```
RickerRDD(rdi, species_params, ...)
```

### Arguments

| | |
|---|---|
| rdi | Vector of density-independent reproduction rates $R_p$ for all species. |
| species_params | A species parameter dataframe. Must contain a column ricker_b holding the coefficient b. |
| ... | Unused |

## Value

Vector of density-dependent reproduction rates.

## See Also

Other functions calculating density-dependent reproduction rate: `BevertonHoltRDD()`, `SheperdRDD()`, `constantRDD()`, `noRDD()`

---

`semichemostat` *Dummy function used during testing only*

---

## Description

Dummy function used during testing only

## Usage

```
semichemostat(params, n_other, rates, dt, component, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `n_other` | A list with the abundances of other components |
| `rates` | A list of rates as returned by `mizerRates()` |
| `dt` | Time step |
| `component` | Name of the component that is being updated |
| `...` | Unused |

---

`setColours` *Set line colours to be used in mizer plots*

---

## Description

Set line colours to be used in mizer plots

## Usage

```
setColours(params, colours)

getColours(params)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `colours` | A named list or named vector of line colours. |

## Value

The MizerParams object with updated line colours

## Examples

```
params <- NS_params
params <- setColours(params, list("Cod" = "red", "Haddock" = "#00ff00"))
plotSpectra(params)
getColours(params)
```

---

`setComponent` *Add a dynamical ecosystem component*

---

## Description

By default, mizer models any number of size-resolved consumer species and a single size-resolved resource spectrum. Your model may require additional components, like for example detritus or carrion or multiple resources or .... This function allows you to set up such components.

## Usage

```
setComponent(
  params,
  component,
  initial_value,
  dynamics_fun,
  encounter_fun,
  mort_fun,
  component_params
)

removeComponent(params, component)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `component` | Name of the component |
| `initial_value` | Initial value of the component |
| `dynamics_fun` | Name of function to calculate value at the next time step |
| `encounter_fun` | Name of function to calculate contribution to encounter rate. Optional. |
| `mort_fun` | Name of function to calculate contribution to the mortality rate. Optional. |
| `component_params` | Named list of parameters needed by the component functions. Optional. |

### Details

The component can be a number, a vector, an array, a list, or any other data structure you like.

If you set a component with a new name, the new component will be added to the existing components. If you set a component with an existing name, that component will be overwritten. You can remove a component with `removeComponent()`.

### Value

The updated MizerParams object

---

## `setExtMort` *Set external mortality rate*

---

### Description

Set external mortality rate

### Usage

```
setExtMort(params, z0 = NULL, z0pre = 0.6, z0exp = -1/4, ...)

getExtMort(params)
```

### Arguments

| | |
|---|---|
| `params` | MizerParams |
| `z0` | Optional. An array (species x size) holding the external mortality rate. |
| `z0pre` | If `z0`, the mortality from other sources, is not a column in the species data frame, it is calculated as z0pre * w_inf ^ z0exp. Default value is 0.6. |
| `z0exp` | If `z0`, the mortality from other sources, is not a column in the species data frame, it is calculated as `z0pre * w_inf ^ z0exp`. Default value is `n-1`. |
| `...` | Unused |

### Value

MizerParams object with updated external mortality rate. Because of the way the R language works, `setExtMort()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setExtMort(params,...)`.

### Setting external mortality rate

The external mortality is all the mortality that is not due to fishing or predation by predators included in the model. The external mortality could be due to predation by predators that are not explicitly included in the model (e.g. mammals or seabirds) or due to other causes like illness. It is a rate with units 1/year.

The `z0` argument allows you to specify an external mortality rate that depends on species and body size. You can see an example of this in the Examples section of the help page for `setExtMort()`.

If the `z0` argument is not supplied, then the external mortality is assumed to depend only on the species, not on the size of the individual: $\mu_{b.i}(w) = z_{0.i}$. The value of the constant $z_0$ for each species is taken from the `z0` column of the species_params data frame, if that column exists. Otherwise it is calculated as

$$z_{0.i} = \texttt{z0pre}_i \, w_{inf}^{\texttt{z0exp}}.$$

## See Also

Other functions for setting parameters: `setFishing()`, `setInitialValues()`, `setInteraction()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setPredKernel()`, `setReproduction()`, `setResource()`, `setSearchVolume()`, `species_params()`

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params)

#### Setting allometric death rate ######################

# Set coefficient for each species. Here we choose 0.1 for each species
z0pre <- rep(0.1, nrow(params@species_params))

# Multiply by power of size with exponent, here chosen to be -1/4
# The outer() function makes it an array species x size
z0 <- outer(z0pre, params@w^(-1/4))

# Change the external mortality rate in the params object
params <- setExtMort(params, z0 = z0)

## End(Not run)
```

---

`setFishing` *Set fishing parameters*

---

## Description

Set fishing parameters

## Usage

```
setFishing(
  params,
  selectivity = NULL,
  catchability = NULL,
  initial_effort = NULL,
  ...
)

gear_params(params)

gear_params(params) <- value

getCatchability(params)

getSelectivity(params)

getInitialEffort(params)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| selectivity | An array (gear x species x size) that holds the selectivity of each gear for species and size, $S_{g,i,w}$. |
| catchability | An array (gear x species) that holds the catchability of each species by each gear, $Q_{g,i}$. |
| initial_effort | Optional. A number or a named numeric vector specifying the fishing effort. If a number, the same effort is used for all gears. If a vector, must be named by gear. |
| ... | Unused |
| value | A data frame with the gear parameters |

## Value

MizerParams object with updated catchability and selectivity. Because of the way the R language works, `setFishing()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setFishing(params,...)`.

## Setting fishing

### Gears

In `mizer`, fishing mortality is imposed on species by fishing gears. The total per-capita fishing mortality (1/year) is obtained by summing over the mortality from all gears,

$$\mu_{f.i}(w) = \sum_g F_{g,i}(w),$$

where the fishing mortality $F_{g,i}(w)$ imposed by gear $g$ on species $i$ at size $w$ is calculated as:

$$F_{g,i}(w) = S_{g,i}(w) Q_{g,i} E_g,$$

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.

**Selectivity**

The selectivity at size of each gear for each species is saved as a three dimensional array (gear x species x size). Each entry has a range between 0 (that gear is not selecting that species at that size) to 1 (that gear is selecting all individuals of that species of that size). This three dimensional array can be specified explicitly via the `selectivity` argument, but usually mizer calculates it from the `gear_params` slot of the MizerParams object.

To allow the calculation of the `selectivity` array, the `gear_params` slot must be a data frame with one row for each gear-species combination. So if for example a gear can select three species, then that gear contributes three rows to the `gear_params` data frame, one for each species it can select. The data frame must have columns `gear`, holding the name of the gear, `species`, holding the name of the species, and `sel_func`, holding the name of the function that calculates the selectivity curve. Some selectivity functions are included in the package: `knife_edge()`, `sigmoid_length()`, `double_sigmoid_length()`, and `sigmoid_weight()`. 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.

Each selectivity function may have parameters. Values for these parameters must be included as columns in the gear parameters data.frame. The names of the columns must exactly match the names of the corresponding arguments of the selectivity function. For example, the default selectivity function is `knife_edge()` that a has sudden change of selectivity from 0 to 1 at a certain size. In its help page you can see that the `knife_edge()` function has arguments `w` and `knife_edge_size`. 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 gear parameters data.frame. So for the `knife_edge()` function there should be a `knife_edge_size` column. Because `knife_edge()` is the default selectivity function, the `knife_edge_size` argument has a default value = `w_mat`.

In case each species is only selected by one gear, the columns of the `gear_params` data frame can alternatively be provided as columns of the `species_params` data frame, if this is more convenient for the user to set up. Mizer will then copy these columns over to create the `gear_params` data frame when it creates the MizerParams object. However changing these columns in the species parameter data frame later will not update the `gear_params` data frame.

**Catchability**

Catchability is used as an additional factor 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.

Catchability is stored as a two dimensional array (gear x species). This can either be provided explicitly via the `catchability` argument, or the information can be provided via a `catchability` column in the `gear_params` data frame.

In the case where each species is selected by only a single gear, the `catchability` column can also be provided in the `species_params` data frame. Mizer will then copy this over to the `gear_params` data frame when the MizerParams object is created.

### Effort

The initial fishing effort is stored in the MizerParams object. If it is not supplied, it is set to zero. The initial effort can be overruled when the simulation is run with project(), where it is also possible to specify an effort that varies through time.

## See Also

Other functions for setting parameters: setExtMort(), setInitialValues(), setInteraction(), setMaxIntakeRate(), setMetabolicRate(), setPredKernel(), setReproduction(), setResource(), setSearchVolume(), species_params()

## Examples

```
## Not run:
params <- NS_params
# Change knife edge size for species 1
params@species_params$knife_edge_size[1] <- 15
params <- setFishing(params)

## End(Not run)
```

---

# setInitialValues *Set initial values to final values of a simulation*

---

## Description

Takes the final values from a simulation in a MizerSim object and stores them as initial values in a MizerParams object.

## Usage

```
setInitialValues(params, sim)
```

## Arguments

| | |
|---|---|
| params | A MizerParams() object |
| sim | A MizerSim object. |

## Value

The params object with updated initial values and initial effort, taken from the values at the final time of the simulation in sim. Because of the way the R language works, setInitialValues() does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form params <-setInitialValues(params,sim).

## See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInteraction()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setPredKernel()`, `setReproduction()`, `setResource()`, `setSearchVolume()`, `species_params()`

## Examples

```
## Not run:
params <- NS_params
sim <- project(params, t_max = 20, effort = 0.5)
params <- setInitialValues(params, sim)

## End(Not run)
```

---

# setInteraction *Set species interaction matrix*

---

## Description

Set species interaction matrix

## Usage

```
setInteraction(params, interaction = NULL)

getInteraction(params)
```

## Arguments

| | |
|---|---|
| `params` | MizerParams object |
| `interaction` | Optional interaction matrix of the species (predator species x prey species). Entries should be numbers between 0 and 1. By default all entries are 1. See "Setting interactions" section below. |

## Value

MizerParams object with updated interaction matrix. Because of the way the R language works, `setInteraction()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setInteraction(params,...)`.

## Setting interactions

The interaction matrix $\theta_{ij}$ 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 encountered food and predation mortality (see on the website the section on predator-prey encounter rate and on predation mortality).

It is used when calculating the food encounter rate in `getEncounter()` and the predation mortality rate in `getPredMort()`. Its entries are dimensionless numbers.The values are between 0 (species do not overlap and therefore do not interact with 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.

This function checks that the supplied interaction matrix is valid and then stores it in the `interaction` slot of the params object before returning that object.

The order of the columns and rows of the `interaction` argument should be the same as the order in the species params data frame in the `params` object. If you supply a named array then the function will check the order and warn if it is different. One way of creating your own interaction matrix is to enter the data using a spreadsheet program and saving it as a .csv file. The data can be read into R using the command `read.csv()`.

The interaction of the species with the resource are set via a column `interaction_resource` in the `species_params` data frame. Again the entries have to be numbers between 0 and 1. By default this column is set to all 1s.

## See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInitialValues()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setPredKernel()`, `setReproduction()`, `setResource()`, `setSearchVolume()`, `species_params()`

## Examples

```
## Not run:
params <- newTraitParams()
interaction <- params@interaction
interaction[1, 3] <- 0
params <- setInteraction(params, interaction)

## End(Not run)
```

---

`setLinetypes` *Set linetypes to be used in mizer plots*

---

## Description

Set linetypes to be used in mizer plots

## Usage

```
setLinetypes(params, linetypes)

getLinetypes(params)
```

### Arguments

| | |
|---|---|
| params | A MizerParams object |
| linetypes | A named list or named vector of linetypes. |

### Value

The MizerParams object with updated linetypes

### Examples

```
params <- NS_params
params <- setLinetypes(params, list("Cod" = "solid"))
plotSpectra(params)
getLinetypes(params)
```

---

## setMaxIntakeRate *Set maximum intake rate*

---

### Description

Set maximum intake rate

### Usage

```
setMaxIntakeRate(params, intake_max = NULL, ...)

getMaxIntakeRate(params)
```

### Arguments

| | |
|---|---|
| params | MizerParams |
| intake_max | Optional. An array (species x size) holding the maximum intake rate for each species at size. If not supplied, a default is set as described in the section "Setting maximum intake rate". |
| ... | Unused |

### Value

A `MizerParams` object with updated maximum intake rate. Because of the way the R language works, `setMaxIntakeRate()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setMaxIntakeRate(params,...)`.

### Setting maximum intake rate

The maximum intake rate $h_i(w)$ of an individual of species $i$ and weight $w$ determines the feeding level, calculated with `getFeedingLevel()`. It is measured in grams/year.

If the `intake_max` argument is not supplied, then the maximum intake rate is set to

$$h_i(w) = h_i w_i^n.$$

The values of $h_i$ (the maximum intake rate of an individual of size 1 gram) and $n_i$ (the allometric exponent for the intake rate) are taken from the `h` and `n` columns in the species parameter dataframe. If the `h` column is not supplied in the species parameter dataframe, it is calculated by the `get_h_default()` function, using `f0` and the `k_vb` column, if they are supplied.

If $h_i$ is set to `Inf`, fish will consume all encountered food.

### See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInitialValues()`, `setInteraction()`, `setMetabolicRate()`, `setPredKernel()`, `setReproduction()`, `setResource()`, `setSearchVolume()`, `species_params()`

### Examples

```
## Not run:
params <- NS_params
params@species_params$h[3] <- 35
params <- setMaxIntakeRate(params)

## End(Not run)
```

---

## setMetabolicRate *Set metabolic rate*

---

### Description

Sets the rate at which energy is used for metabolism and activity

### Usage

```
setMetabolicRate(params, metab = NULL, p = NULL, ...)

getMetabolicRate(params)
```

### Arguments

| | |
|---|---|
| `params` | MizerParams |
| `metab` | Optional. An array (species x size) holding the metabolic rate for each species at size. If not supplied, a default is set as described in the section "Setting metabolic rate". |

| | |
|---|---|
| p | The allometric metabolic exponent. This is only used if metab is not given explicitly and if the exponent is not specified in a p column in the species_params. |
| ... | Unused |

## Value

MizerParams object with updated metabolic rate. Because of the way the R language works, setMetabolicRate() does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form params <-setMetabolicRate(params,...).

## Setting metabolic rate

The metabolic rate is subtracted from the energy income rate to calculate the rate at which energy is available for growth and reproduction, see getEReproAndGrowth(). It is measured in grams/year.

If the metab argument is not supplied, then for each species the metabolic rate $k(w)$ for an individual of size $w$ is set to

$$k(w) = ksw^p + kw,$$

where $ksw^p$ represents the rate of standard metabolism and $kw$ is the rate at which energy is expended on activity and movement. The values of $ks$, $p$ and $k$ are taken from the ks, p and k columns in the species parameter dataframe. If any of these parameters are not supplied, the defaults are $k = 0$, $p = n$ and

$$ks = f_c h \alpha w_{mat}^{n-p},$$

where $f_c$ is the critical feeding level taken from the fc column in the species parameter data frame. If the critical feeding level is not specified, a default of $f_c = 0.2$ is used.

## See Also

Other functions for setting parameters: setExtMort(), setFishing(), setInitialValues(), setInteraction(), setMaxIntakeRate(), setPredKernel(), setReproduction(), setResource(), setSearchVolume(), species_params()

## Examples

```
## Not run:
params <- NS_params
# Change activity coefficient for species 3
params@species_params$k[3] <- 8
params <- setMetabolicRate(params)

## End(Not run)
```

`setPredKernel` *Set predation kernel*

## Description

The predation kernel determines the distribution of prey sizes that a predator feeds on. It is used in `getEncounter()` when calculating the rate at which food is encountered and in `getPredRate()` when calculating the rate at which a prey is predated upon. The predation kernel can be a function of the predator/prey size ratio or it can be a function of the predator size and the prey size separately. Both types can be set up with this function.

## Usage

```
setPredKernel(params, pred_kernel = NULL, ...)
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `pred_kernel` | Optional. An array (species x predator size x prey size) that holds the predation coefficient of each predator at size on each prey size. If not supplied, a default is set as described in section "Setting predation kernel". |
| `...` | Unused |

## Value

A MizerParams object with updated predation kernel. Because of the way the R language works, `setPredKernel()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setPredKernel(params,...)`.

## Setting predation kernel

### Kernel dependent on predator to prey size ratio

If the `pred_kernel` argument is not supplied, then this function sets a predation kernel that depends only on the ratio of predator mass to prey mass, not on the two masses independently. The shape of that kernel is then determined by the `pred_kernel_type` column in species_params.

The default pred_kernel_type is "lognormal". This will call the function `lognormal_pred_kernel()` to calculate the predation kernel. An alternative pred_kernel type is "box", implemented by the function `box_pred_kernel()`, and "power_law", implemented by the function `power_law_pred_kernel()`. These functions require certain species parameters in the species_params data frame. For the log-normal kernel these are `beta` and `sigma`, for the box kernel they are `ppmr_min` and `ppmr_max`. They are explained in the help pages for the kernel functions. Except for `beta` and `sigma`, no defaults are set for these parameters. If they are missing from the species_params data frame then mizer will issue an error message.

You can use any other string as the type. If for example you choose "my" then you need to define a function `my_pred_kernel` that you can model on the existing functions like `lognormal_pred_kernel()`.

When using a kernel that depends on the predator/prey size ratio only, mizer does not need to store the entire three dimensional array in the MizerParams object. Such an array can be very big when there is a large number of size bins. Instead, mizer only needs to store two two-dimensional arrays that hold Fourier transforms of the feeding kernel function that allow the encounter rate and the predation rate to be calculated very efficiently. However, if you need the full three-dimensional array you can calculate it with the `getPredKernel()` function.

**Kernel dependent on both predator and prey size**

If you want to work with a feeding kernel that depends on predator mass and prey mass independently, you can specify the full feeding kernel as a three-dimensional array (predator species x predator size x prey size).

You should use this option only if a kernel dependent only on the predator/prey mass ratio is not appropriate. Using a kernel dependent on predator/prey mass ratio only allows mizer to use fast Fourier transform methods to significantly reduce the running time of simulations.

The order of the predator species in `pred_kernel` should be the same as the order in the species params dataframe in the `params` object. If you supply a named array then the function will check the order and warn if it is different.

## See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInitialValues()`, `setInteraction()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setReproduction()`, `setResource()`, `setSearchVolume()`, `species_params()`

## Examples

```
## Not run:
## Set up a MizerParams object
params <- newMultispeciesParams(NS_species_params_gears, inter)

## If you change predation kernel parameters after setting up a model, you
# need to call setPredKernel
params@species_params["Cod", "beta"] <- 200
params <- setPredKernel(params)

## You can change to a different predation kernel type
params@species_params$pred_kernel_type <- "box"
params@species_params$ppmr_min <- 2
params@species_params$ppmr_max <- 4
params <- setPredKernel(params)

## If you need a kernel that depends also on prey size you need to define
# it yourself.
pred_kernel <- getPredKernel(params)
pred_kernel["Herring", , ] <- sweep(pred_kernel["Herring", , ], 2,
                                    params@w_full, "*")
params<- setPredKernel(params, pred_kernel = pred_kernel)

## End(Not run)
```

`setRateFunction` *Set own rate function to replace mizer rate function*

## Description

If the way mizer calculates a fundamental rate entering the model is not flexible enough for you (for example if you need to introduce time dependence) then you can write your own functions for calculating that rate and use `setRateFunction()` to register it with mizer.

## Usage

```
setRateFunction(params, rate, fun)

getRateFunction(params, rate)

other_params(params)

other_params(params) <- value
```

## Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `rate` | Name of the rate for which a new function is to be set. |
| `fun` | Name of the function to use to calculate the rate. |
| `value` | Values for other parameters |

## Details

At each time step during a simulation with the `project()` function, mizer needs to calculate the instantaneous values of the various rates. By default it calls the `mizerRates()` function which creates a list with the following components:

- `encounter` from `mizerEncounter()`
- `feeding_level` from `mizerFeedingLevel()`
- `pred_rate` from `mizerPredRate()`
- `pred_mort` from `mizerPredMort()`
- `fishing_mort` from `mizerFMort()`
- `mort` from `mizerMort()`
- `resource_mort` from `mizerResourceMort()`
- `e` from `mizerEReproAndGrowth()`
- `e_repro` from `mizerERepro()`
- `e_growth` from `mizerEGrowth()`
- `rdi` from `mizerRDI()`

- rdd from BevertonHoltRDD()

For each of these you can substitute your own function. So for example if you have written your own function for calculating the total mortality rate and have called it myMort and have a mizer model stored in a MizerParams object called params that you want to run with your new mortality rate, then you would call

```
params <- setRateFunction(params, "Mort", "myMort")
```

In general if you want to replace a function mizerSomeRateFunc() with a function myVersionOfThis() you would call

```
params <- setRateFunction(params, "SomeRateFunc", "myVersionOfThis")
```

In some extreme cases you may need to swap out the entire mizerRates() function for your own function called myRates(). That you can do with

```
params <- setRateFunction(params, "Rates", "myRates")
```

Your new rate functions may need their own model parameters. These you can store in other_params(params). For example

```
other_params(params)$my_param <- 42
```

Note that your own rate functions need to be defined in the global environment or in a package. If they are defined within a function then mizer will not find them.

## Value

For setRateFunction(): An updated MizerParams object

For getRateFunction(): The name of the registered rate function for the requested rate, or the list of all rate functions if called without rate argument.

For other_params(): A named list with all the parameters for which you have set values.

---

setReproduction *Set reproduction parameters*

---

## Description

Sets the proportion of the total energy available for reproduction and growth that is invested into reproduction as a function of the size of the individual and sets additional density dependence.

## Usage

```
setReproduction(params, maturity = NULL, repro_prop = NULL, RDD = NULL, ...)

getMaturityProportion(params)

getReproductionProportion(params)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| maturity | Optional. An array (species x size) that holds the proportion of individuals of each species at size that are mature. If not supplied, a default is set as described in the section "Setting reproduction". |
| repro_prop | Optional. An array (species x size) that holds the proportion of consumed energy that a mature individual allocates to reproduction for each species at size. If not supplied, a default is set as described in the section "Setting reproduction". |
| RDD | The name of the function calculating the density-dependent reproduction rate from the density-independent rate. Defaults to "BevertonHoltRDD()". |
| ... | Unused |

## Value

The updated MizerParams object. Because of the way the R language works, setReproduction() does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form params <-setReproduction(params,...).

## Setting reproduction

**Investment:** For each species and at each size, the proportion $\psi$ of the available energy that is invested into reproduction is the product of two factors: the proportion maturity of individuals that are mature and the proportion repro_prop of the energy available to a mature individual that is invested into reproduction.

If the maturity argument is not supplied, then it is set to a sigmoidal maturity ogive that changes from 0 to 1 at around the maturity size:

$$\texttt{maturity}(w) = \left[1 + \left(\frac{w}{w_{mat}}\right)^{-U}\right]^{-1}.$$

(To avoid clutter, we are not showing the species index in the equations.) The maturity weights are taken from the w_mat column of the species_params data frame. Any missing maturity weights are set to 1/4 of the asymptotic weight in the w_inf column. The exponent $U$ determines the steepness of the maturity ogive. By default it is chosen as $U = 10$, however this can be overridden by including a column w_mat25 in the species parameter dataframe that specifies the weight at which 25% of individuals are mature, which sets $U = \log(3)/\log(w_{mat}/w_{25})$.

The sigmoidal function given above would strictly reach 1 only asymptotically. Mizer instead sets the function equal to 1 already at the species' maximum size, taken from the compulsory w_inf column in the species_params data frame.

If the repro_prop argument is not supplied, it is set to the allometric form

$$\texttt{repro\_prop}(w) = \left(\frac{w}{w_{inf}}\right)^{m-n}.$$

Here $n$ is the scaling exponent of the energy income rate. Hence the exponent $m$ determines the scaling of the investment into reproduction for mature individuals. By default it is chosen to be $m = 1$ so that the rate at which energy is invested into reproduction scales linearly with the size.

This default can be overridden by including a column `m` in the species parameter dataframe. The asymptotic sizes are taken from the compulsory `w_inf` column in the species_params data frame. So finally we have

$$\psi(w) = \texttt{maturity}(w)\texttt{repro\_prop}(w)$$

**Efficiency:** The reproductive efficiency, i.e., the proportion of energy allocated to reproduction that results in egg biomass, is set through the `erepro` column in the species_params data frame. If that is not provided, the default is set to 1 (which you will want to override). The offspring biomass divided by the egg biomass gives the rate of egg production, returned by `getRDI()`.

**Density dependence:** The stock-recruitment relationship is an emergent phenomenon in mizer, with several sources of density dependence. Firstly, the amount of energy invested into reproduction depends on the energy income of the spawners, which is density-dependent due to competition for prey. Secondly, the proportion of larvae that grow up to recruitment size depends on the larval mortality, which depends on the density of predators, and on larval growth rate, which depends on density of prey.

Finally, to encode all the density dependence in the stock-recruitment relationship that is not already included in the other two sources of density dependence, mizer puts the the density-independent rate of egg production through a density-dependence function. The result is returned by `getRDD()`. The name of the density-dependence function is specified by the `RDD` argument. The default is the Beverton-Holt function `BevertonHoltRDD()`, which requires an `R_max` column in the species_params data frame giving the maximum egg production rate. If this column does not exist, it is initialised to `Inf`, leading to no density-dependence. Other functions provided by mizer are `RickerRDD()` and `ShepherdRDD()` and you can easily use these as models for writing your own functions.

## See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInitialValues()`, `setInteraction()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setPredKernel()`, `setResource()`, `setSearchVolume()`, `species_params()`

## Examples

```
## Not run:
params <- NS_params
# Change maturity size for species 3
params@species_params$w_mat[3] <- 24
params <- setReproduction(params)

## End(Not run)
```

---

`setResource` *Set up resource*

---

## Description

Sets the intrinsic resource growth rate and the intrinsic resource carrying capacity as well as the name of the function used to simulate the resource dynamics

## Usage

```
setResource(
  params,
  resource_rate = NULL,
  resource_capacity = NULL,
  r_pp = resource_params(params)[["r_pp"]],
  kappa = resource_params(params)[["kappa"]],
  lambda = resource_params(params)[["lambda"]],
  n = resource_params(params)[["n"]],
  w_pp_cutoff = resource_params(params)[["w_pp_cutoff"]],
  resource_dynamics = NULL,
  ...
)

getResourceRate(params)

getResourceCapacity(params)

getResourceDynamics(params)

resource_params(params)

resource_params(params) <- value
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| resource_rate | Optional. Vector of resource intrinsic birth rates |
| resource_capacity | Optional. Vector of resource intrinsic carrying capacity |
| r_pp | Coefficient of the intrinsic resource birth rate |
| kappa | Coefficient of the intrinsic resource carrying capacity |
| lambda | Scaling exponent of the intrinsic resource carrying capacity |
| n | Allometric growth exponent for resource |
| w_pp_cutoff | The upper cut off size of the resource spectrum. Default is 10 g. |
| resource_dynamics | Function that determines resource dynamics by calculating the resource spectrum at the next time step from the current state. |
| ... | Unused |
| value | List of resource parameters |

## Value

A MizerParams object with updated resource parameters. Because of the way the R language works, `setResource()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setResource(params,...)`.

### Setting resource dynamics

By default, mizer uses a semichemostat model to describe the resource dynamics in each size class independently. This semichemostat dynamics is implemented by the function `resource_semichemostat()`. You can change the resource dynamics by writing your own function, modelled on `resource_semichemostat()`, and then passing the name of your function in the `resource_dynamics` argument.

The `resource_rate` argument is a vector specifying the intrinsic resource growth rate for each size class. If it is not supplied, then the intrinsic growth rate $r(w)$ at size $w$ is set to

$$r(w) = r_{pp}\, w^{n-1}.$$

The values of $r_{pp}$ and $n$ are taken from the `r_pp` and `n` arguments.

The `resource_capacity` argument is a vector specifying the intrinsic resource carrying capacity for each size class. If it is not supplied, then the intrinsic carrying capacity $c(w)$ at size $w$ is set to

$$c(w) = \kappa\, w^{-\lambda}$$

for all $w$ less than `w_pp_cutoff` and zero for larger sizes. The values of $\kappa$ and $\lambda$ are taken from the `kappa` and `lambda` arguments.

### See Also

Other functions for setting parameters: `setExtMort()`, `setFishing()`, `setInitialValues()`, `setInteraction()`, `setMaxIntakeRate()`, `setMetabolicRate()`, `setPredKernel()`, `setReproduction()`, `setSearchVolume()`, `species_params()`

---

## setRmax *Set maximum reproduction rate*

---

### Description

Takes a MizerParams object with density-independent reproduction rate and sets a Beverton-Holt density-dependence with a maximum reproduction rate that is a chosen factor `R_factor` higher than the initial-state reproduction rate. At the same time it adjust the reproductive efficiency `erepro` (see `setReproduction()`) to keep the same density-dependent reproduction at the initial state.

### Usage

```
setRmax(params, R_factor)
```

### Arguments

| | |
|---|---|
| `params` | A MizerParams object |
| `R_factor` | The factor by which the maximum reproduction rate should be higher than the initial-state reproduction rate |

### Value

A MizerParams object

`setSearchVolume` *Set search volume*

## Description

Set search volume

## Usage

```
setSearchVolume(params, search_vol = NULL, ...)

getSearchVolume(params)
```

## Arguments

| | |
|---|---|
| `params` | MizerParams |
| `search_vol` | Optional. An array (species x size) holding the search volume for each species at size. If not supplied, a default is set as described in the section "Setting search volume". |
| `...` | Unused |

## Value

MizerParams with updated search volume. Because of the way the R language works, `setSearchVolume()` does not make the changes to the params object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setSearchVolume(params,...)`.

## Setting search volume

The search volume $\gamma_i(w)$ of an individual of species $i$ and weight $w$ multiplies the predation kernel when calculating the encounter rate in `getEncounter()` and the predation rate in `getPredRate()`.

The name "search volume" is a bit misleading, because $\gamma_i(w)$ does not have units of volume. It is simply a parameter that determines the rate of predation. Its units depend on your choice, see section "Units in mizer". If you have chose to work with total abundances, then it is a rate with units 1/year. If you have chosen to work with abundances per m^2 then it has units of m^2/year. If you have chosen to work with abundances per m^3 then it has units of m^3/year.

If the `search_vol` argument is not supplied, then the search volume is set to

$$\gamma_i(w) = \gamma_i w_i^q.$$

The values of $\gamma_i$ (the search volume at 1g) and $q_i$ (the allometric exponent of the search volume) are taken from the `gamma` and `q` columns in the species parameter dataframe. If the `gamma` column is not supplied in the species parameter dataframe, a default is calculated by the `get_gamma_default()` function. Note that only for predators of size $w = 1$ gram is the value of the species parameter $\gamma_i$ the same as the value of the search volume $\gamma_i(w)$.

## See Also

Other functions for setting parameters: setExtMort(), setFishing(), setInitialValues(), setInteraction(), setMaxIntakeRate(), setMetabolicRate(), setPredKernel(), setReproduction(), setResource(), species_params()

## Examples

```
## Not run:
params <- newTraitParams()
params@species_params$gamma[3] <- 1000
params <- setSearchVolume(params)

## End(Not run)
```

---

# set_community_model *Deprecated function for setting up parameters for a community-type model*

---

## Description

This function has been deprecated in favour of the function newCommunityParams() that sets better default values.

## Usage

```
set_community_model(
  max_w = 1e+06,
  min_w = 0.001,
  min_w_pp = 1e-10,
  z0 = 0.1,
  alpha = 0.2,
  h = 10,
  beta = 100,
  sigma = 2,
  q = 0.8,
  n = 2/3,
  kappa = 1000,
  lambda = 2 + q - n,
  f0 = 0.7,
  r_pp = 10,
  gamma = NA,
  knife_edge_size = 1000,
  knife_is_min = TRUE,
  recruitment = kappa * min_w^-lambda,
  rec_mult = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| max_w | The maximum size of the community. The w_inf of the species used to represent the community is set to this value. The default value is 1e6. |
| min_w | The minimum size of the community. Default value is 1e-3. |
| min_w_pp | The smallest size of the resource spectrum. |
| z0 | The background mortality of the community. Default value is 0.1. |
| alpha | The assimilation efficiency of the community. Default value 0.2 |
| h | The maximum food intake rate. Default value is 10. |
| beta | The preferred predator prey mass ratio. Default value is 100. |
| sigma | The width of the prey preference. Default value is 2.0. |
| q | The search volume exponent. Default value is 0.8. |
| n | The scaling of the intake. Default value is 2/3. |
| kappa | The carrying capacity of the resource spectrum. Default value is 1000. |
| lambda | The exponent of the resource spectrum. Default value is 2 + q<br>• n. |
| f0 | The average feeding level of individuals who feed on a power-law spectrum. This value is used to calculate the search rate parameter gamma (see the package vignette). Default value is 0.7. |
| r_pp | Growth rate parameter for the resource spectrum. Default value is 10. |
| gamma | Volumetric search rate. Estimated using h, f0 and kappa if not supplied. |
| knife_edge_size | The size at the edge of the knife-selectivity function. Default value is 1000. |
| knife_is_min | Is the knife-edge selectivity function selecting above (TRUE) or below (FALSE) the edge. Default is TRUE. |
| recruitment | The constant recruitment in the smallest size class of the community spectrum. This should be set so that the community spectrum continues the resource spectrum. Default value = kappa * min_w^-lambda. |
| rec_mult | Additional multiplier for the constant recruitment. Default value is 1. |
| ... | Other arguments to pass to the MizerParams constructor. |

## Details

This functions creates a MizerParams object so that community-type models can be easily set up and run. A community model has several features that distinguish it from the food-web type models. Only one 'species' is resolved, i.e. one 'species' is used to represent the whole community. The resource spectrum only extends to the start of the community spectrum. Recruitment to the smallest size in the community spectrum is constant and set by the user. As recruitment is constant, the proportion of energy invested in reproduction (the slot psi of the returned MizerParams object) is set to 0. Standard metabolism has been turned off (the parameter ks is set to 0). Consequently, the growth rate is now determined solely by the assimilated food (see the package vignette for more details).

The function has many arguments, all of which have default values. The main arguments that the users should be concerned with are `z0`, `recruitment`, `alpha` and `f0` as these determine the average growth rate of the community.

Fishing selectivity is modelled as a knife-edge function with one parameter, `knife_edge_size`, which determines the size at which species are selected.

The resulting `MizerParams` object can be projected forward using `project()` like any other `MizerParams` object. When projecting the community model it may be necessary to keep a small time step size `dt` of around 0.1 to avoid any instabilities with the solver. You can check for these numerical instabilities by plotting the biomass or abundance through time after the projection.

## Value

An object of type `MizerParams`

## References

K. H. Andersen,J. E. Beyer and P. Lundberg, 2009, Trophic and individual efficiencies of size-structured communities, Proceedings of the Royal Society, 276, 109-114

## See Also

Other deprecated functions: `set_multispecies_model()`, `set_trait_model()`

## Examples

```
## Not run:
params <- set_community_model(f0=0.7, z0=0.2, recruitment=3e7)
sim <- project(params, effort = 0, t_max = 100, dt=0.1)
plotBiomass(sim)
plotSpectra(sim)

## End(Not run)
```

---

# set_multispecies_model

*Deprecated obsolete function for setting up multispecies parameters*

---

## Description

This function has been deprecated in favour of the function `newMultispeciesParams()` that sets better default values.

## Usage

```
set_multispecies_model(
  species_params,
 interaction = matrix(1, nrow = nrow(species_params), ncol = nrow(species_params)),
  min_w_pp = 1e-10,
  min_w = 0.001,
  max_w = max(species_params$w_inf) * 1.1,
  no_w = 100,
  n = 2/3,
  q = 0.8,
  f0 = 0.6,
  kappa = 1e+11,
  lambda = 2 + q - n,
  r_pp = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| species_params | A data frame of species-specific parameter values. |
| interaction | Optional interaction matrix of the species (predator species x prey species). Entries should be numbers between 0 and 1. By default all entries are 1. See "Setting interactions" section below. |
| min_w_pp | The smallest size of the resource spectrum. By default this is set to the smallest value at which any of the consumers can feed. |
| min_w | Sets the size of the eggs of all species for which this is not given in the w_min column of the species_params dataframe. |
| max_w | The largest size of the consumer spectrum. By default this is set to the largest w_inf specified in the species_params data frame. |
| no_w | The number of size bins in the consumer spectrum. |
| n | The allometric growth exponent. This can be overruled for individual species by including a n column in the species_params. |
| q | Allometric exponent of search volume |
| f0 | Expected average feeding level. Used to set gamma, the coefficient in the search rate. Ignored if gamma is given explicitly. |
| kappa | Coefficient of the intrinsic resource carrying capacity |
| lambda | Scaling exponent of the intrinsic resource carrying capacity |
| r_pp | Coefficient of the intrinsic resource birth rate |
| ... | Unused |

## See Also

Other deprecated functions: set_community_model(), set_trait_model()

`set_trait_model` *Deprecated function for setting up parameters for a trait-based model*

## Description

This function has been deprecated in favour of the function `newTraitParams()` that sets better default values.

## Usage

```
set_trait_model(
  no_sp = 10,
  min_w_inf = 10,
  max_w_inf = 1e+05,
  no_w = 100,
  min_w = 0.001,
  max_w = max_w_inf * 1.1,
  min_w_pp = 1e-10,
  w_pp_cutoff = 1,
  k0 = 50,
  n = 2/3,
  p = 0.75,
  q = 0.9,
  eta = 0.25,
  r_pp = 4,
  kappa = 0.005,
  lambda = 2 + q - n,
  alpha = 0.6,
  ks = 4,
  z0pre = 0.6,
  h = 30,
  beta = 100,
  sigma = 1.3,
  f0 = 0.5,
  gamma = NA,
  knife_edge_size = 1000,
  gear_names = "knife_edge_gear",
  ...
)
```

## Arguments

| | |
|---|---|
| `no_sp` | The number of species in the model. The default value is 10. The more species, the longer takes to run. |
| `min_w_inf` | The asymptotic size of the smallest species in the community. |
| `max_w_inf` | The asymptotic size of the largest species in the community. |

| | |
|---|---|
| no_w | The number of size bins in the community spectrum. |
| min_w | The smallest size of the community spectrum. |
| max_w | Obsolete argument because the maximum size of the consumer spectrum is set to max_w_inf. |
| min_w_pp | Obsolete argument because the smallest resource size is set to the smallest size at which the consumers feed. |
| w_pp_cutoff | The cut off size of the resource spectrum. Default value is 1. |
| k0 | Multiplier for the maximum recruitment. Default value is 50. |
| n | Scaling of the intake. Default value is 2/3. |
| p | Scaling of the standard metabolism. Default value is 0.75. |
| q | Exponent of the search volume. Default value is 0.9. |
| eta | Factor to calculate w_mat from asymptotic size. |
| r_pp | Growth rate parameter for the resource spectrum. Default value is 4. |
| kappa | Coefficient in abundance power law. Default value is 0.005. |
| lambda | Exponent of the abundance power law. Default value is (2+q-n). |
| alpha | The assimilation efficiency of the community. The default value is 0.6 |
| ks | Standard metabolism coefficient. Default value is 4. |
| z0pre | The coefficient of the background mortality of the community. z0 = z0pre * w_inf ^ (n-1). The default value is 0.6. |
| h | Maximum food intake rate. Default value is 30. |
| beta | Preferred predator prey mass ratio. Default value is 100. |
| sigma | Width of prey size preference. Default value is 1.3. |
| f0 | Expected average feeding level. Used to set gamma, the factor for the search volume. The default value is 0.5. |
| gamma | Volumetric search rate. Estimated using h, f0 and kappa if not supplied. |
| knife_edge_size | The minimum size at which the gear or gears select species. Must be of length 1 or no_sp. |
| gear_names | The names of the fishing gears. A character vector, the same length as the number of species. Default is 1 - no_sp. |
| ... | Other arguments to pass to the MizerParams constructor. |

## Details

This functions creates a MizerParams object so that trait-based-type models can be easily set up and run. The trait-based size spectrum model can be derived as a simplification of the general size-based model used in mizer. The species-specific parameters are the same for all species, except for the asymptotic size, which is considered the most important trait characterizing a species. Other parameters are related to the asymptotic size. For example, the size at maturity is given by w_inf * eta, where eta is the same for all species. For the trait-based model the number of species is not important. For applications of the trait-based model see Andersen & Pedersen (2010). See the mizer vignette for more details and examples of the trait-based model.

The function has many arguments, all of which have default values. Of particular interest to the user are the number of species in the model and the minimum and maximum asymptotic sizes. The asymptotic sizes of the species are spread evenly on a logarithmic scale within this range.

The stock recruitment relationship is the default Beverton-Holt style. The maximum recruitment is calculated using equilibrium theory (see Andersen & Pedersen, 2010) and a multiplier, `k0`. Users should adjust `k0` to get the spectra they want.

The factor for the search volume, `gamma`, is calculated using the expected feeding level, `f0`.

Fishing selectivity is modelled as a knife-edge function with one parameter, `knife_edge_size`, which is the size at which species are selected. Each species can either be fished by the same gear (`knife_edge_size` has a length of 1) or by a different gear (the length of `knife_edge_size` has the same length as the number of species and the order of selectivity size is that of the asymptotic size).

The resulting `MizerParams` object can be projected forward using `project` like any other `MizerParams` object. When projecting the community model it may be necessary to reduce `dt` to 0.1 to avoid any instabilities with the solver. You can check this by plotting the biomass or abundance through time after the projection.

## Value

An object of type `MizerParams`

## References

K. H. Andersen and M. Pedersen, 2010, Damped trophic cascades driven by fishing in model marine ecosystems. Proceedings of the Royal Society V, Biological Sciences, 1682, 795-802.

## See Also

Other deprecated functions: `set_community_model()`, `set_multispecies_model()`

---

`SheperdRDD` *Sheperd function to calculate density-dependent reproduction rate*

---

## Description

Takes the density-independent rates $R_p$ of egg production and returns reduced, density-dependent rates $R$ as

$$R = \frac{R_p}{1 + (b\, R_p)^c}$$

## Usage

```
SheperdRDD(rdi, species_params, ...)
```

### Arguments

| | |
|---|---|
| rdi | Vector of density-independent reproduction rates $R_p$ for all species. |
| species_params | A species parameter dataframe. Must contain columns sheperd_b and sheperd_c with the parameters b and c. |
| ... | Unused |

### Value

Vector of density-dependent reproduction rates.

### See Also

Other functions calculating density-dependent reproduction rate: BevertonHoltRDD(), RickerRDD(), constantRDD(), noRDD()

---

## sigmoid_length *Length based sigmoid selectivity function*

---

### Description

A sigmoid shaped selectivity function. Based on two parameters l25 and l50 which determine the length at which 25% and 50% of the stock is selected respectively. As the size-based model is weight based, and this selectivity function is length based, it uses the length-weight parameters a and b to convert between length and weight.

### Usage

```
sigmoid_length(w, l25, l50, species_params, ...)
```

### Arguments

| | |
|---|---|
| w | the size of the individual. |
| l25 | the length which gives a selectivity of 25%. |
| l50 | the length which gives a selectivity of 50%. |
| species_params | A list with the species params for the current species. Used to get at the length-weight parameters a and b |
| ... | Unused |

## sigmoid_weight *Weight based sigmoidal selectivity function*

### Description

A sigmoidal selectivity function with 50% selectivity at weight `sigmoidal_weight` and width `sigmoidal_sigma`.

### Usage

```
sigmoid_weight(w, sigmoidal_weight, sigmoidal_sigma, ...)
```

### Arguments

| | |
|---|---|
| `w` | The size of the individual. |
| `sigmoidal_weight` | The weight at which the knife-edge operates. |
| `sigmoidal_sigma` | The width of the selection function |
| `...` | Unused |

## species_params *Species parameters*

### Description

This is the right place to document the use of species parameters in mizer.

This is a convenient wrapper function calling each of the following functions

- `setPredKernel()`
- `setSearchVolume()`
- `setInteraction()`
- `setMaxIntakeRate()`
- `setMetabolicRate()`
- `setExtMort()`
- `setReproduction()`
- `setFishing()`
- `setResource()`

See the Details section below for a discussion of how to use this function.

## Usage

```
species_params(params)

species_params(params) <- value

setParams(params, interaction = NULL, ...)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| value | A data frame with the species parameters |
| interaction | Optional interaction matrix of the species (predator species x prey species). Entries should be numbers between 0 and 1. By default all entries are 1. See "Setting interactions" section below. |
| ... | Arguments passed on to setPredKernel, setSearchVolume, setMaxIntakeRate, setMetabolicRate, setExtMort, setReproduction, setFishing, setResource |

pred_kernel Optional. An array (species x predator size x prey size) that holds the predation coefficient of each predator at size on each prey size. If not supplied, a default is set as described in section "Setting predation kernel".

search_vol Optional. An array (species x size) holding the search volume for each species at size. If not supplied, a default is set as described in the section "Setting search volume".

intake_max Optional. An array (species x size) holding the maximum intake rate for each species at size. If not supplied, a default is set as described in the section "Setting maximum intake rate".

metab Optional. An array (species x size) holding the metabolic rate for each species at size. If not supplied, a default is set as described in the section "Setting metabolic rate".

p The allometric metabolic exponent. This is only used if metab is not given explicitly and if the exponent is not specified in a p column in the species_params.

z0 Optional. An array (species x size) holding the external mortality rate.

z0pre If z0, the mortality from other sources, is not a column in the species data frame, it is calculated as z0pre * w_inf ^ z0exp. Default value is 0.6.

z0exp If z0, the mortality from other sources, is not a column in the species data frame, it is calculated as z0pre * w_inf ^ z0exp. Default value is n-1.

maturity Optional. An array (species x size) that holds the proportion of individuals of each species at size that are mature. If not supplied, a default is set as described in the section "Setting reproduction".

repro_prop Optional. An array (species x size) that holds the proportion of consumed energy that a mature individual allocates to reproduction for each species at size. If not supplied, a default is set as described in the section "Setting reproduction".

RDD The name of the function calculating the density-dependent reproduction rate from the density-independent rate. Defaults to "BevertonHoltRDD()".

selectivity An array (gear x species x size) that holds the selectivity of each gear for species and size, $S_{g,i,w}$.

`catchability` An array (gear x species) that holds the catchability of each species by each gear, $Q_{g,i}$.

`initial_effort` Optional. A number or a named numeric vector specifying the fishing effort. If a number, the same effort is used for all gears. If a vector, must be named by gear.

`resource_rate` Optional. Vector of resource intrinsic birth rates

`resource_capacity` Optional. Vector of resource intrinsic carrying capacity

`r_pp` Coefficient of the intrinsic resource birth rate

`n` Allometric growth exponent for resource

`kappa` Coefficient of the intrinsic resource carrying capacity

`lambda` Scaling exponent of the intrinsic resource carrying capacity

`w_pp_cutoff` The upper cut off size of the resource spectrum. Default is 10 g.

`resource_dynamics` Function that determines resource dynamics by calculating the resource spectrum at the next time step from the current state.

## Details

Usually, if you are happy with the way mizer calculates its model functions from the species parameters and only want to change the values of some species parameters, you would make those changes in the `species_params` data frame contained in the `params` object and then call the `setParams()` function to effect the change. Note that just changing the species parameters by themselves is not changing the model until you call `setParams()` or the appropriate one of its sub-functions. Here is an example which assumes that you have have a MizerParams object `params` in which you just want to change one parameter of the third species:

```
params@species_params$gamma[[3]] <- 1000
params <- setParams(params)
```

Because of the way the R language works, `setParams` does not make the changes to the `params` object that you pass to it but instead returns a new params object. So to affect the change you call the function in the form `params <-setParams(params,...)`.

If you are not happy with the assumptions that mizer makes by default about the shape of the model functions, for example if you want to change one of the allometric scaling assumptions, you can do this by providing your choice as an array in the appropriate argument to `setParams()`. The sections below discuss all the model functions that you can change this way.

This function will use the species parameters in the `params` object to reset the values of all the model functions that you do not specify explicitly when calling this function, unless you have protected the corresponding slots with a comment. If you have changed any of the model functions in the `params` object previously and now want to make changes to a different slot, you will want to call the appropriate change function individually. So in the above example you would have used `params <-setSearchVolume(params)` instead of `params <-setParams(params)`.

If you have added a comment to a slot of the params object, then setParams() and its subfunctions will not recalculate the value for that slot from the species parameters. For example

```
comment(params@search_vol) <- "This should not change"
species_params(params)$gamma <- 10
```

will just issue a warning "The search volume has been commented and therefore will not be recalculated from the species parameters". You can remove the comment, and therefore allow recalculation of the slot, with `comment(params@search_vol) <-NULL`.

## Value

A MizerParams object

## Units in mizer

Mizer uses grams to measure weight, centimetres to measure lengths, and years to measure time.

Mizer is agnostic about whether abundances are given as

1. numbers per area,
2. numbers per volume or
3. total numbers for the entire study area.

You should make the choice most convenient for your application and then stick with it. If you make choice 1 or 2 you will also have to choose a unit for area or volume. Your choice will then determine the units for some of the parameters. This will be mentioned when the parameters are discussed in the sections below.

You choice will also affect the units of the quantities you may want to calculate with the model. For example, the yield will be in grams/year/m^2 in case 1 if you choose m^2 as your measure of area, in grams/year/m^3 in case 2 if you choose m^3 as your unit of volume, or simply grams/year in case 3. The same comment applies for other measures, like total biomass, which will be grams/area in case 1, grams/volume in case 2 or simply grams in case 3. When mizer puts units on axes, for example in `plotBiomass`, it will simply put grams, as appropriate for case 3.

You can convert between these choices. For example, if you use case 1, you need to multiply with the area of the ecosystem to get the total quantity. If you work with case 2, you need to multiply by both area and the thickness of the productive layer. In that respect, case 2 is a bit cumbersome.

## Setting interactions

The interaction matrix $\theta_{ij}$ 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 encountered food and predation mortality (see on the website the section on predator-prey encounter rate and on predation mortality).

It is used when calculating the food encounter rate in `getEncounter()` and the predation mortality rate in `getPredMort()`. Its entries are dimensionless numbers.The values are between 0 (species do not overlap and therefore do not interact with 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.

This function checks that the supplied interaction matrix is valid and then stores it in the `interaction` slot of the params object before returning that object.

The order of the columns and rows of the `interaction` argument should be the same as the order in the species params data frame in the `params` object. If you supply a named array then the function

will check the order and warn if it is different. One way of creating your own interaction matrix is to enter the data using a spreadsheet program and saving it as a .csv file. The data can be read into R using the command `read.csv()`.

The interaction of the species with the resource are set via a column `interaction_resource` in the `species_params` data frame. Again the entries have to be numbers between 0 and 1. By default this column is set to all 1s.

## Setting predation kernel

### Kernel dependent on predator to prey size ratio

If the `pred_kernel` argument is not supplied, then this function sets a predation kernel that depends only on the ratio of predator mass to prey mass, not on the two masses independently. The shape of that kernel is then determined by the `pred_kernel_type` column in species_params.

The default pred_kernel_type is "lognormal". This will call the function `lognormal_pred_kernel()` to calculate the predation kernel. An alternative pred_kernel type is "box", implemented by the function `box_pred_kernel()`, and "power_law", implemented by the function `power_law_pred_kernel()`. These functions require certain species parameters in the species_params data frame. For the log-normal kernel these are `beta` and `sigma`, for the box kernel they are `ppmr_min` and `ppmr_max`. They are explained in the help pages for the kernel functions. Except for `beta` and `sigma`, no defaults are set for these parameters. If they are missing from the species_params data frame then mizer will issue an error message.

You can use any other string as the type. If for example you choose "my" then you need to define a function `my_pred_kernel` that you can model on the existing functions like `lognormal_pred_kernel()`.

When using a kernel that depends on the predator/prey size ratio only, mizer does not need to store the entire three dimensional array in the MizerParams object. Such an array can be very big when there is a large number of size bins. Instead, mizer only needs to store two two-dimensional arrays that hold Fourier transforms of the feeding kernel function that allow the encounter rate and the predation rate to be calculated very efficiently. However, if you need the full three-dimensional array you can calculate it with the `getPredKernel()` function.

### Kernel dependent on both predator and prey size

If you want to work with a feeding kernel that depends on predator mass and prey mass independently, you can specify the full feeding kernel as a three-dimensional array (predator species x predator size x prey size).

You should use this option only if a kernel dependent only on the predator/prey mass ratio is not appropriate. Using a kernel dependent on predator/prey mass ratio only allows mizer to use fast Fourier transform methods to significantly reduce the running time of simulations.

The order of the predator species in `pred_kernel` should be the same as the order in the species params dataframe in the `params` object. If you supply a named array then the function will check the order and warn if it is different.

## Setting search volume

The search volume $\gamma_i(w)$ of an individual of species $i$ and weight $w$ multiplies the predation kernel when calculating the encounter rate in `getEncounter()` and the predation rate in `getPredRate()`.

The name "search volume" is a bit misleading, because $\gamma_i(w)$ does not have units of volume. It is simply a parameter that determines the rate of predation. Its units depend on your choice, see

section "Units in mizer". If you have chose to work with total abundances, then it is a rate with units 1/year. If you have chosen to work with abundances per m^2 then it has units of m^2/year. If you have chosen to work with abundances per m^3 then it has units of m^3/year.

If the `search_vol` argument is not supplied, then the search volume is set to

$$\gamma_i(w) = \gamma_i w_i^q.$$

The values of $\gamma_i$ (the search volume at 1g) and $q_i$ (the allometric exponent of the search volume) are taken from the `gamma` and `q` columns in the species parameter dataframe. If the `gamma` column is not supplied in the species parameter dataframe, a default is calculated by the `get_gamma_default()` function. Note that only for predators of size $w = 1$ gram is the value of the species parameter $\gamma_i$ the same as the value of the search volume $\gamma_i(w)$.

### Setting maximum intake rate

The maximum intake rate $h_i(w)$ of an individual of species $i$ and weight $w$ determines the feeding level, calculated with `getFeedingLevel()`. It is measured in grams/year.

If the `intake_max` argument is not supplied, then the maximum intake rate is set to

$$h_i(w) = h_i w_i^n.$$

The values of $h_i$ (the maximum intake rate of an individual of size 1 gram) and $n_i$ (the allometric exponent for the intake rate) are taken from the `h` and `n` columns in the species parameter dataframe. If the `h` column is not supplied in the species parameter dataframe, it is calculated by the `get_h_default()` function, using `f0` and the `k_vb` column, if they are supplied.

If $h_i$ is set to `Inf`, fish will consume all encountered food.

### Setting metabolic rate

The metabolic rate is subtracted from the energy income rate to calculate the rate at which energy is available for growth and reproduction, see `getEReproAndGrowth()`. It is measured in grams/year.

If the `metab` argument is not supplied, then for each species the metabolic rate $k(w)$ for an individual of size $w$ is set to

$$k(w) = ksw^p + kw,$$

where $ksw^p$ represents the rate of standard metabolism and $kw$ is the rate at which energy is expended on activity and movement. The values of $ks$, $p$ and $k$ are taken from the `ks`, `p` and `k` columns in the species parameter dataframe. If any of these parameters are not supplied, the defaults are $k = 0$, $p = n$ and

$$ks = f_c h \alpha w_{mat}^{n-p},$$

where $f_c$ is the critical feeding level taken from the `fc` column in the species parameter data frame. If the critical feeding level is not specified, a default of $f_c = 0.2$ is used.

### Setting external mortality rate

The external mortality is all the mortality that is not due to fishing or predation by predators included in the model. The external mortality could be due to predation by predators that are not explicitly included in the model (e.g. mammals or seabirds) or due to other causes like illness. It is a rate with units 1/year.

The z0 argument allows you to specify an external mortality rate that depends on species and body size. You can see an example of this in the Examples section of the help page for setExtMort().

If the z0 argument is not supplied, then the external mortality is assumed to depend only on the species, not on the size of the individual: $\mu_{b.i}(w) = z_{0.i}$. The value of the constant $z_0$ for each species is taken from the z0 column of the species_params data frame, if that column exists. Otherwise it is calculated as

$$z_{0.i} = \texttt{z0pre}_i \, w_{inf}^{\texttt{z0exp}}.$$

## Setting reproduction

**Investment:** For each species and at each size, the proportion $\psi$ of the available energy that is invested into reproduction is the product of two factors: the proportion maturity of individuals that are mature and the proportion repro_prop of the energy available to a mature individual that is invested into reproduction.

If the maturity argument is not supplied, then it is set to a sigmoidal maturity ogive that changes from 0 to 1 at around the maturity size:

$$\texttt{maturity}(w) = \left[1 + \left(\frac{w}{w_{mat}}\right)^{-U}\right]^{-1}.$$

(To avoid clutter, we are not showing the species index in the equations.) The maturity weights are taken from the w_mat column of the species_params data frame. Any missing maturity weights are set to 1/4 of the asymptotic weight in the w_inf column. The exponent $U$ determines the steepness of the maturity ogive. By default it is chosen as $U = 10$, however this can be overridden by including a column w_mat25 in the species parameter dataframe that specifies the weight at which 25% of individuals are mature, which sets $U = \log(3)/\log(w_{mat}/w_{25})$.

The sigmoidal function given above would strictly reach 1 only asymptotically. Mizer instead sets the function equal to 1 already at the species' maximum size, taken from the compulsory w_inf column in the species_params data frame.

If the repro_prop argument is not supplied, it is set to the allometric form

$$\texttt{repro\_prop}(w) = \left(\frac{w}{w_{inf}}\right)^{m-n}.$$

Here $n$ is the scaling exponent of the energy income rate. Hence the exponent $m$ determines the scaling of the investment into reproduction for mature individuals. By default it is chosen to be $m = 1$ so that the rate at which energy is invested into reproduction scales linearly with the size. This default can be overridden by including a column m in the species parameter dataframe. The asymptotic sizes are taken from the compulsory w_inf column in the species_params data frame.

So finally we have

$$\psi(w) = \texttt{maturity}(w)\texttt{repro\_prop}(w)$$

**Efficiency:** The reproductive efficiency, i.e., the proportion of energy allocated to reproduction that results in egg biomass, is set through the erepro column in the species_params data frame. If that is not provided, the default is set to 1 (which you will want to override). The offspring biomass divided by the egg biomass gives the rate of egg production, returned by getRDI().

**Density dependence:** The stock-recruitment relationship is an emergent phenomenon in mizer, with several sources of density dependence. Firstly, the amount of energy invested into reproduction depends on the energy income of the spawners, which is density-dependent due to competition for prey. Secondly, the proportion of larvae that grow up to recruitment size depends on the larval mortality, which depends on the density of predators, and on larval growth rate, which depends on density of prey.

Finally, to encode all the density dependence in the stock-recruitment relationship that is not already included in the other two sources of density dependence, mizer puts the the density-independent rate of egg production through a density-dependence function. The result is returned by `getRDD()`. The name of the density-dependence function is specified by the `RDD` argument. The default is the Beverton-Holt function `BevertonHoltRDD()`, which requires an `R_max` column in the species_params data frame giving the maximum egg production rate. If this column does not exist, it is initialised to `Inf`, leading to no density-dependence. Other functions provided by mizer are `RickerRDD()` and `SheperdRDD()` and you can easily use these as models for writing your own functions.

## Setting fishing

### Gears

In `mizer`, fishing mortality is imposed on species by fishing gears. The total per-capita fishing mortality (1/year) is obtained by summing over the mortality from all gears,

$$\mu_{f.i}(w) = \sum_g F_{g,i}(w),$$

where the fishing mortality $F_{g,i}(w)$ imposed by gear $g$ on species $i$ at size $w$ is calculated as:

$$F_{g,i}(w) = S_{g,i}(w) Q_{g,i} E_g,$$

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.

### Selectivity

The selectivity at size of each gear for each species is saved as a three dimensional array (gear x species x size). Each entry has a range between 0 (that gear is not selecting that species at that size) to 1 (that gear is selecting all individuals of that species of that size). This three dimensional array can be specified explicitly via the `selectivity` argument, but usually mizer calculates it from the `gear_params` slot of the MizerParams object.

To allow the calculation of the `selectivity` array, the `gear_params` slot must be a data frame with one row for each gear-species combination. So if for example a gear can select three species, then that gear contributes three rows to the `gear_params` data frame, one for each species it can select. The data frame must have columns `gear`, holding the name of the gear, `species`, holding the name of the species, and `sel_func`, holding the name of the function that calculates the selectivity curve. Some selectivity functions are included in the package: `knife_edge()`, `sigmoid_length()`, `double_sigmoid_length()`, and `sigmoid_weight()`. 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.

Each selectivity function may have parameters. Values for these parameters must be included as columns in the gear parameters data.frame. The names of the columns must exactly match the

names of the corresponding arguments of the selectivity function. For example, the default selectivity function is `knife_edge()` that a has sudden change of selectivity from 0 to 1 at a certain size. In its help page you can see that the `knife_edge()` function has arguments `w` and `knife_edge_size`. 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 gear parameters data.frame. So for the `knife_edge()` function there should be a `knife_edge_size` column. Because `knife_edge()` is the default selectivity function, the `knife_edge_size` argument has a default value = `w_mat`.

In case each species is only selected by one gear, the columns of the `gear_params` data frame can alternatively be provided as columns of the `species_params` data frame, if this is more convenient for the user to set up. Mizer will then copy these columns over to create the `gear_params` data frame when it creates the MizerParams object. However changing these columns in the species parameter data frame later will not update the `gear_params` data frame.

**Catchability**

Catchability is used as an additional factor 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.

Catchability is stored as a two dimensional array (gear x species). This can either be provided explicitly via the `catchability` argument, or the information can be provided via a `catchability` column in the `gear_params` data frame.

In the case where each species is selected by only a single gear, the `catchability` column can also be provided in the `species_params` data frame. Mizer will then copy this over to the `gear_params` data frame when the MizerParams object is created.

**Effort**

The initial fishing effort is stored in the `MizerParams` object. If it is not supplied, it is set to zero. The initial effort can be overruled when the simulation is run with `project()`, where it is also possible to specify an effort that varies through time.

## Setting resource dynamics

By default, mizer uses a semichemostat model to describe the resource dynamics in each size class independently. This semichemostat dynamics is implemented by the function `resource_semichemostat()`. You can change the resource dynamics by writing your own function, modelled on `resource_semichemostat()`, and then passing the name of your function in the `resource_dynamics` argument.

The `resource_rate` argument is a vector specifying the intrinsic resource growth rate for each size class. If it is not supplied, then the intrinsic growth rate $r(w)$ at size $w$ is set to

$$r(w) = r_{pp}\, w^{n-1}.$$

The values of $r_{pp}$ and $n$ are taken from the `r_pp` and `n` arguments.

The `resource_capacity` argument is a vector specifying the intrinsic resource carrying capacity for each size class. If it is not supplied, then the intrinsic carrying capacity $c(w)$ at size $w$ is set to

$$c(w) = \kappa\, w^{-\lambda}$$

for all $w$ less than `w_pp_cutoff` and zero for larger sizes. The values of $\kappa$ and $\lambda$ are taken from the `kappa` and `lambda` arguments.

## See Also

Other functions for setting parameters: setExtMort(), setFishing(), setInitialValues(), setInteraction(), setMaxIntakeRate(), setMetabolicRate(), setPredKernel(), setReproduction(), setResource(), setSearchVolume()

## Examples

```
## Not run:
params <- newTraitParams()
params@species_params$gamma[3] <- 1000
params <- setParams(params)

## End(Not run)
```

---

steady *Set initial values to a steady state for the model*

---

## Description

The steady state is found by running the dynamics while keeping reproduction and other components constant until the size spectra no longer change (or until time t_max is reached if earlier) Then the reproductive efficiencies are set to the values that give the level of reproduction observed in that steady state.

## Usage

```
steady(
  params,
  t_max = 100,
  t_per = 1.5,
  tol = 10^(-2),
  dt = 0.1,
  return_sim = FALSE,
  progress_bar = TRUE
)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| t_max | The maximum number of years to run the simulation. Default is 100. |
| t_per | The simulation is broken up into shorter runs of t_per years, after each of which we check for convergence. Default value is 1.5. This should be chosen as an odd multiple of the timestep dt in order to be able to detect period 2 cycles. |
| tol | The simulation stops when the relative change in the egg production RDI over t_per years is less than tol for every species. Default value is 1/100. |
| dt | The time step to use in project(). |

| | |
|---|---|
| return_sim | If TRUE, the function returns the MizerSim object holding the result of the simulation run. If FALSE (default) the function returns a MizerParams object with the "initial" slots set to the steady state. |
| progress_bar | A shiny progress object to implement a progress bar in a shiny app. Default FALSE. |

## Examples

```
## Not run:
params <- newTraitParams()
params@species_params$gamma[5] <- 3000
params <- setSearchVolume(params)
params <- steady(params)
plotSpectra(params)

## End(Not run)
```

---

summary,MizerParams-method

*Summarize MizerParams object*

---

## Description

Outputs a general summary of the structure and content of the object

## Usage

```
## S4 method for signature 'MizerParams'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | A MizerParams object. |
| ... | Other arguments (currently not used). |

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears,inter)
summary(params)

## End(Not run)
```

summary,MizerSim-method *Summarize MizerSim object*

## Description

Outputs a general summary of the structure and content of the object

## Usage

```
## S4 method for signature 'MizerSim'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | A MizerSim object. |
| ... | Other arguments (currently not used). |

## Examples

```
## Not run:
params <- newMultispeciesParams(NS_species_params_gears,inter)
sim <- project(params, effort=1, t_max=5)
summary(sim)

## End(Not run)
```

summary_functions *Description of summary functions*

## Description

Mizer provides a range of functions to summarise the results of a simulation.

## Details

A list of available summary functions is given in the table below.

| Function | Returns | Description |
|---|---|---|
| getDiet() | Three dimensional array (predator x size x prey) | Diet of predator at size, resolved by prey speci |
| getSSB() | Two dimensional array (time x species) | Total Spawning Stock Biomass (SSB) of each |
| 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 t |
| getM2 | Three dimensional array (time x species x size) | The predation mortality imposed on each spec |
| getFMort() | Three dimensional array (time x species x size) | Total fishing mortality on each species by size |

| | | |
|---|---|---|
| getFMortGear() | Four dimensional array (time x gear x species x size) | Fishing mortality on each species by each gea |
| 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 thr |

## See Also

indicator_functions, plotting_functions

---

`test_dyn` *Dummy function used during testing only*

---

## Description

Dummy function used during testing only

## Usage

```
test_dyn(params, ...)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |
| ... | Other parameters |

---

`truncated_lognormal_pred_kernel`
*Truncated lognormal predation kernel*

---

## Description

This is like the lognormal_pred_kernel() but with an imposed maximum predator/prey mass ratio

## Usage

```
truncated_lognormal_pred_kernel(ppmr, beta, sigma)
```

## Arguments

| | |
|---|---|
| ppmr | A vector of predator/prey size ratios |
| beta | The preferred predator/prey size ratio |
| sigma | The width parameter of the log-normal kernel |

## Details

Writing the predator mass as $w$ and the prey mass as $w_p$, the feeding kernel is given as

$$\phi_i(w, w_p) = \exp\left[\frac{-(\ln(w/w_p/\beta_i))^2}{2\sigma_i^2}\right]$$

if $w/w_p$ is between 1 and $\beta_i \exp(3\sigma_i)$ and zero otherwise. Here $\beta_i$ is the preferred predator-prey mass ratio and $\sigma_i$ determines the width of the kernel. These two parameters need to be given in the species parameter dataframe in the columns `beta` and `sigma`.

This function is called from `setPredKernel()` to set up the predation kernel slots in a MizerParams object.

## Value

A vector giving the value of the predation kernel at each of the predator/prey mass ratios in the `ppmr` argument.

---

`upgradeParams` *Upgrade MizerParams object from earlier mizer versions*

---

## Description

Occasionally during the development of new features for mizer, the MizerParams object gains extra slots. MizerParams objects created in older versions of mizer are then no longer valid in the new version because of the missing slots. You need to upgrade them with

```
params <- upgradeParams(params)
```

where `params` should be replaced by the name of your MizerParams object. This function adds the missing slots and fills them with default values. Any object from version 0.4 onwards can be upgraded. Any old MizerSim objects should be similarly updated with `upgradeSim()`. This function uses `newMultispeciesParams()` to create a new MizerParams object using the parameters extracted from the old MizerParams object.

## Usage

```
upgradeParams(params)
```

## Arguments

| | |
|---|---|
| `params` | An old MizerParams object to be upgraded |

## Value

The upgraded MizerParams object

### Backwards compatibility

The internal numerics in mizer have changed over time, so there may be small discrepancies between the results obtained with the upgraded object in the new version and the original object in the old version. If it is important for you to reproduce the exact results then you should install the version of mizer with which you obtained the results. You can do this with

```
remotes::install_github("sizespectrum/mizer", ref = "v0.2")
```

where you should replace "v0.2" with the version number you require. You can see the list of available releases at https://github.com/sizespectrum/mizer/tags.

If you only have a serialised version of the old object, for example created via `saveRDS()`, and you get an error when trying to read it in with `readRDS()` then unfortunately you will need to install the old version of mizer first to read the params object into your workspace, then switch to the current version and then call `upgradeParams()`. You can then save the new version again with `saveRDS()`.

---

## upgradeSim *Upgrade MizerSim object from earlier mizer versions*

---

### Description

Occasionally, during the development of new features for mizer, the MizerSim class or the MizerParams class gains extra slots. MizerSim objects created in older versions of mizer are then no longer valid in the new version because of the missing slots. You need to upgrade them with

```
sim <- upgradeSim(sim)
```

where `sim` should be replaced by the name of your MizerSim object.

### Usage

```
upgradeSim(sim)
```

### Arguments

| | |
|---|---|
| `sim` | An old MizerSim object to be upgraded |

### Details

This function adds the missing slots and fills them with default values. It calls `upgradeParams()` to upgrade the MizerParams object inside the MizerSim object. Any object from version 0.4 onwards can be upgraded.

### Value

The upgraded MizerSim object

### Backwards compatibility

The internal numerics in mizer have changed over time, so there may be small discrepancies between the results obtained with the upgraded object in the new version and the original object in the old version. If it is important for you to reproduce the exact results then you should install the version of mizer with which you obtained the results. You can do this with

```
remotes::install_github("sizespectrum/mizer", ref = "v0.2")
```

where you should replace "v0.2" with the version number you require. You can see the list of available releases at https://github.com/sizespectrum/mizer/tags.

If you only have a serialised version of the old object, for example created via `saveRDS()`, and you get an error when trying to read it in with `readRDS()` then unfortunately you will need to install the old version of mizer first to read the params object into your workspace, then switch to the current version and then call `upgradeParams()`. You can then save the new version again with `saveRDS()`.

---

## validGearParams *Check validity of gear parameters and set defaults*

---

### Description

The function returns a valid gear parameter data frame that can be used by `setFishing()` or it gives an error message.

### Usage

```
validGearParams(gear_params, species_params)
```

### Arguments

| | |
|---|---|
| `gear_params` | Gear parameter data frame |
| `species_params` | Species parameter data frame |

### Details

If gear_params is empty, then this function tries to find the necessary information in the species_params data frame. This restricts each species to be fished by only one gear. Defaults are used for information that can not be found in the species_params dataframe, as follows:

- If there is no `gear` column, each species gets its own gear, named after the species.
- If there is no `sel_func` column or it is NA then `knife_edge` is used.
- If there is no `catchability` column or it is NA then this is set to 1.
- If the selectivity function is `knife_edge` and no `knife_edge_size` is provided, it is set to `w_mat`.

### Value

A valid gear parameter data frame

## validSpeciesParams *Validate species parameter data frame*

### Description

Check validity of species parameters and set defaults for missing but required parameters

### Usage

```
validSpeciesParams(species_params)
```

### Arguments

| | |
|---|---|
| species_params | The user-supplied species parameter data frame |

### Value

A valid species parameter data frame

This function throws an error if

- the species column does not exist or contains duplicates
- the w_inf column does not exist or contains NAs or is not numeric

It sets default values if any of the following are missing or NA

- w_mat is set to w_inf/4
- w_min is set to 0.001
- alpha is set to 0.6
- interaction_resource is set to 1

Any w_mat25 that is given that is not smaller than w_mat is set to w_mat * 3^(-0.1).

If species_params was provided as a tibble it is converted back to an ordinary data frame.

## w *Size bins*

### Description

Functions to fetch information about the size bins used in the model described by params.

## Usage

```
w(params)

w_full(params)

dw(params)

dw_full(params)
```

## Arguments

| | |
|---|---|
| params | A MizerParams object |

## Details

TODO: Give more details about how mizer discretises the size

## Value

`w()` returns a vector with the sizes at the start of each size bin of the community spectrum.

`w_full()` returns a vector with the sizes at the start of each size bin of the resource spectrum, which typically starts at smaller sizes than the community spectrum.

`dw()` returns a vector with the widths of the size bins of the community spectrum.

`dw_full()` returns a vector with the widths of the size bins of the resource spectrum.

# Index