

Package: FIMS (via r-universe)

October 17, 2026

Title The Fisheries Integrated Modeling System

Version 0.9.2

Description The Fisheries Integrated Modeling System is a next-generation framework of stock assessment models, assisting fishery managers with the goal of achieving sustainable fisheries. This system, when completed in a few years, offers the NOAA Fisheries and global fisheries science communities an advanced set of stock assessment models. These tools can be used separately or in combination to incorporate ecosystem and socioeconomic data and models, as well as climate effects and other drivers within the marine environment, into stock assessment models.

License GPL (>= 3) | file LICENSE

URL <https://github.com/noaa-fims/fims>, <https://noaa-fims.github.io>,
<https://noaa-fisheries-integrated-toolbox.r-universe.dev/FIMS>,
<https://noaa-fims.github.io/FIMS>

BugReports <https://github.com/noaa-fims/fims/issues>

Depends R (>= 4.1.0)

Imports cli, dplyr, ggplot2, glue, graphics, jsonlite, methods, purrr,
Rcpp, rlang, scales, stats, tibble, TMB (>= 1.8.0), utils

Suggests covr, devtools, DT, htmltools, knitr, lozen, markdown,
microbenchmark, parallel, pkgdown, remotes, rmarkdown,
snowfall, SparseNUTS, spelling, stockplotr, styler, testdown,
testthat (>= 3.0.0), tidyr, usethis, withr

LinkingTo Rcpp, RcppEigen, TMB

VignetteBuilder knitr

Remotes github::nmfs-ost/stockplotr, github::noaa-afsc/SparseNUTS,
github::ThinkR-open/lozen, github::ThinkR-open/testdown

Additional_repositories <https://noaa-fisheries-integrated-toolbox.r-universe.dev>

Config/testthat/edition 3

Config/testthat/parallel false
Encoding UTF-8
Language en-US
LazyData true
NeedsCompilation yes
Roxygen list(markdown = TRUE)
RoxygenNote 7.3.3
SystemRequirements GNU make
Repository <https://andrea-havron-noaa.r-universe.dev>
Date/Publication 2026-04-02 10:03:30 UTC
RemoteUrl <https://github.com/Andrea-Havron-NOAA/FIMS>
RemoteRef HEAD
RemoteSha cbcbaae7c15e2fb2c034a9a21f24a9796660dcf1

Contents

Cpp_classes	3
Cpp_functions	4
create_default_configurations	5
create_default_parameters	7
data_big	9
fims_input_types	10
FIMSFit	10
FIMSFrame	12
fit_fims	13
get_FIMSFit	14
get_FIMSFrame	16
initialize_data_distribution	19
initialize_fims	21
is.FIMSFit	22
is_fims_verbose	22
lognormal	23
model_	24
plot,FIMSFrame,missing-method	26
remove_test_data	27
reshape_json_estimates	27
reshape_tmb_estimates	28
run_gtest	28
run_r_integration_tests	30
run_r_unit_tests	30
setup_and_run_gtest	31
setup_gtest	31
use_gtest_template	32
use_testthat_template	33

Description

The following classes are exported from C++ via Rcpp. Their documentation is provided in the rendered C++ documentation. For detailed usage, see the C++ docs.

Details

- [AgeComp](#)
- [BevertonHoltRecruitment](#)
- [CatchAtAge](#)
- [DlnormDistribution](#)
- [DmultinomDistribution](#)
- [DnormDistribution](#)
- [DoubleLogisticSelectivity](#)
- [EWAAGrowth](#)
- [Fleet](#)
- [Index](#)
- [Landings](#)
- [LengthComp](#)
- [LogDevsRecruitmentProcess](#)
- [LogRRecruitmentProcess](#)
- [LogisticMaturity](#)
- [LogisticSelectivity](#)
- [Parameter](#)
- [ParameterVector](#)
- [Population](#)
- [RealVector](#)
- [SharedInt](#)
- [SharedReal](#)
- [SharedString](#)

C++ Documentation

[NOAA-FIMS C++ Documentation](#)

Cpp_functions

C++ Functions Exported via Rcpp

Description

The following functions are exported from C++ via Rcpp. Their documentation is provided in the rendered C++ documentation. For detailed usage, see the C++ docs.

Details

- [clear](#)
- [get_fixed](#)
- [get_log](#)
- [get_log_errors](#)
- [get_log_warnings](#)
- [get_parameter_names](#)
- [get_random](#)
- [get_random_names](#)
- [inv_logit](#)
- [log_error](#)
- [log_info](#)
- [log_warning](#)
- [logit](#)
- [set_fixed](#)
- [set_log_throw_on_error](#)
- [set_random](#)
- [CreateTMBModel](#)

C++ Documentation

[NOAA-FIMS C++ Documentation](#)

create_default_configurations

Create a default FIMS configuration tibble

Description

A FIMS configuration tibble specifies the high-level model configuration, i.e., what parameter sets are needed, for a FIMS model. This high-level information can then be passed to `create_default_parameters()`, which returns a tibble with all of the parameters required to run your specified configuration. For example, logistic selectivity is the default, and this function will specify logistic selectivity for every fleet it finds in data. If you want double-normal selectivity you can run this function first and then edit the tibble that is returned, changing the entry in the `module_type` column before running `create_default_parameters()`. Thus, running this function is not strictly necessary to run a FIMS model but it is helpful to see what parameter sets are necessary given the data that you want to model. It automatically creates configuration entries for data modules (e.g., landings, index, compositions) and, depending on the model family, standard population dynamics modules (recruitment, growth, maturity) and selectivity modules for fleets.

Usage

```
create_default_configurations(data, model_family = c("catch_at_age"))
```

Arguments

<code>data</code>	A <code>FIMSFrame</code> object returned from running <code>FIMSFrame()</code> on your long input data.
<code>model_family</code>	A string specifying the model family. Defaults to "catch_at_age".

Details

The function inspects the data to find unique combinations of fleet names and data types. It then maps these to the appropriate FIMS module names and joins them with a predefined template of default settings. When the `model_family` is "catch_at_age", it also adds default configurations for:

- **Selectivity:** A logistic selectivity module for each unique fleet.
- **Recruitment:** A Beverton–Holt recruitment module.
- **Growth:** An empirical weight-at-age (EWAA) growth module.
- **Maturity:** A logistic maturity module.

The final output is a nested tibble, which serves as a starting point for building a complete FIMS model configuration. You can unnest the rows to see the full configuration with `tidyr::unnest()`, see the examples below for more details.

Value

A nested tibble with the following top-level columns is returned:

model_family: The specified model family (e.g., "catch_at_age").

module_name: The name of the FIMS module (e.g., "Data", "Selectivity", "Recruitment", "Growth", "Maturity"). These entries are always written in PascalCase to match the names used in the C++ code.

fleet_name: The name of the fleet the module applies to. This will be NA for non-fleet-specific modules like "Recruitment".

data: A list-column containing a tibble with detailed configurations. Unnesting this column reveals:

module_type: The specific type of the module (e.g., "Logistic" for a "Selectivity" module). This column will always be written in PascalCase to match the names used in the C++ code.

distribution_type: The type of distribution (e.g., "Data", "process"), where a process distribution can refer to a fixed effect or a random effect but it does not fit to data, e.g., recruitment deviations.

distribution: The name of distribution (e.g., "Dlnorm", "Dmultinom"). The column will always be written in PascalCase to match the names used in the C++ code.

See Also

- [FIMSFrame\(\)](#)
- [create_default_parameters\(\)](#)

Examples

```
# Load the example dataset and create a FIMS data frame
data("data_big")
fims_frame <- FIMSFrame(data_big)

# Create the default model configuration tibble
default_configurations <- create_default_configurations(data = fims_frame)

# Unnest the data column to see detailed configurations
default_configurations_unnest <- default_configurations |>
  tidyr::unnest(cols = data) |>
  print()

# Use dplyr::rows_update to modify the selectivity specified for fleet1
# from logistic (the default) to double logistic
configurations_double_logistic <- default_configurations_unnest |>
  dplyr::rows_update(
    tibble::tibble(
      module_name = "Selectivity",
      fleet_name = "fleet1",
      module_type = "DoubleLogistic"
    ),
    by = c("module_name", "fleet_name")
  )
```

```
) |>
print()
```

```
create_default_parameters
```

Create default parameters for a FIMS model

Description

This function generates a tibble with all of the parameters necessary to run a FIMS model given the desired high-level configuration that is specified in `configurations`. The tibble contains default initial values and estimation settings required to build and run the model. You can edit the returned tibble if you want to change things such as initial values to values more specific to your population before running your model. For example, the default maturity parameters will need modified.

Usage

```
create_default_parameters(configurations, data)
```

Arguments

<code>configurations</code>	A tibble of model configurations. Typically created by <code>create_default_configurations()</code> . Users can modify this tibble to customize the model structure before using it as input to this function.
<code>data</code>	A <code>FIMSFrame</code> object returned from running <code>FIMSFrame()</code> on your long input data.

Details

The function processes the `configurations` tibble, which only contains high-level information for running your model by calling internal helper functions on each row and returning a multi-row parameter set for each input row. For example, if a selectivity for the first fleet is configured as "Logistic", it takes that single row of input information and returns a parameter set with two rows, one for each parameter, "inflection_point" and "slope".

Value

A nested tibble containing information on parameters for your model with the same top-level columns as the input tibble passed to `configurations` but with additional information in the nested data column. See below for more details:

`model_family`: The specified model family (e.g., "catch_at_age").

`module_name`: The name of the FIMS module (e.g., "Data", "Selectivity", "Recruitment", "Growth", "Maturity"). These entries are always written in PascalCase to match the names used in the C++ code.

`fleet_name`: The name of the fleet the module applies to. This will be NA for non-fleet-specific modules like "Recruitment".

data: A list-column containing a tibble with detailed parameters. Unnesting this column reveals:

- module_type:** The specific type of the module (e.g., "Logistic" for a "Selectivity" module). This column will always be written in PascalCase to match the names used in the C++ code.
- label:** The name of the parameter (e.g., "inflection_point").
- age:** The age the parameter applies to.
- length:** The length bin the parameter applies to.
- time:** The time step (i.e., year) the parameter applies to.
- value:** The initial value of the parameter.
- estimation_type:** The type of estimation (e.g., "constant", "fixed_effects", "random_effects").
- distribution_type:** The type of distribution (e.g., "Data", "process"), where a process distribution can refer to a fixed effect or a random effect but it does not fit to data, e.g., recruitment deviations.
- distribution:** The name of distribution (e.g., "Dlnorm", "Dmultinom"). The column will always be written in PascalCase to match the names used in the C++ code.

See Also

- [FIMSFrame\(\)](#)
- [create_default_configurations\(\)](#)

Examples

```
## Not run:
# Load the example dataset and create a FIMS data frame
data("data_big")
fims_frame <- FIMSFrame(data_big)

# Create default configurations
default_configurations <- create_default_configurations(fims_frame)

# Create default parameters
default_parameters <- create_default_parameters(
  configurations = default_configurations,
  data = fims_frame
) |>
tidyr::unnest(cols = data)

# Update selectivity parameters for survey1
updated_parameters <- default_parameters |>
dplyr::rows_update(
  tibble::tibble(
    fleet_name = "survey1",
    label = c("inflection_point", "slope"),
    value = c(1.5, 2)
  ),
  by = c("fleet_name", "label")
)
```

```

# Do the same as above except, model fleet1 with double logistic selectivity
# To see required parameters for double logistic selectivity, run
# show(DoubleLogisticSelectivity) and look at the Fields list
parameters_with_double_logistic <- default_configurations |>
  tidyr::unnest(cols = data) |>
  dplyr::rows_update(
    tibble::tibble(
      module_name = "Selectivity",
      fleet_name = "fleet1",
      module_type = "DoubleLogistic"
    ),
    by = c("module_name", "fleet_name")
  ) |>
  create_default_parameters(
    data = fims_frame
  )

## End(Not run)

```

data_big

FIMS input data frame

Description

A data frame containing the information needed to run an age-structured stock assessment model in FIMS. This data was generated using the ASSAMC package written for the [model comparison project](#). The source code to make this dataset is in data-raw/data_big.R which is on GitHub but will not be in your local installation because it is in the .Rbuildignore file.

Usage

```
data_big
```

Format

A data frame with 2808 observations of 8 variables:

type The type of input the row contains. Allowed types include age_comp, age_to_length_conversion, index, landings, length_comp, and weight_at_age.

name A character string providing the name of the information source that the input was collected from, e.g., "Trawl fishery".

age An integer age. Entry can be NA if information pertains to multiple ages, e.g., total landings rather than landings of age-4 fish. In this dataset, ages start at one, therefore recruitment happens at age one not age zero.

length A numeric length. Entry can be NA if information doesn't pertain to length.

timing The timing, i.e., year the data was collected.

value The measurement of interest.

- unit** A character string specifying the units of value. Allowed units for each data type are as follows. `mt` is used for `index`, `landings`, and `weight_at_age` data. `number` or `proportion` are each viable units for the composition data, where the former is the preferred unit of measurement.
- uncertainty** A real value providing a measurement of uncertainty for value. For `landings` and `indices of abundance` this should be the standard deviation of the logged observations if you are using the lognormal distribution to fit your data. For composition data it will be your input sample size.

Source

www.github.com/NOAA-FIMS/Age_Structured_Stock_Assessment_Model_Comparison

<code>fims_input_types</code>	<i>Vector of acceptable FIMS input types</i>
-------------------------------	--

Description

A vector of strings that define which input types are allowed in the column labeled `type` in the data slot of a `FIMSFrame` object. Strings are composed of words separated by underscores, e.g., `"weight_at_age"`.

Usage

`fims_input_types`

Format

A vector of strings.

<code>FIMSFit</code>	<i>Class constructors for class FIMSFit and associated child classes</i>
----------------------	--

Description

Create an object with the class of `FIMSFit` after running a FIMS model. This is typically done within `fit_fims()` but it can be create manually by the user if they have used their own bespoke code to fit a FIMS model.

Usage

```
FIMSFit(
  input,
  obj,
  opt = list(),
  sdreport = list(),
  timing = c(time_total = as.difftime(0, units = "secs")),
  version = utils::packageVersion("FIMS")
)
```

Arguments

<code>input</code>	Input list as returned by <code>initialize_fims()</code> .
<code>obj</code>	An object returned from <code>TMB::MakeADFun()</code> .
<code>opt</code>	An object returned from an optimizer, typically from <code>stats::nlminb()</code> , used to fit a TMB model.
<code>sdreport</code>	An object of the <code>sdreport</code> class as returned from <code>TMB::sdreport()</code> .
<code>timing</code>	A vector of at least length one, where all entries are of the <code>timediff</code> class and at least one is named "time_total". This information is available in <code>fit_fims()</code> and added to this argument internally but if you are a power user you can calculate the time it took to run your model by subtracting two <code>Sys.time()</code> objects.
<code>version</code>	The version of FIMS that was used to optimize the model. If <code>fit_fims()</code> was not used to optimize the model, then the default is to use the current version of the package that is loaded.

Value

An object with an S4 class of `FIMSFit` is returned. The object will have the following slots:

`input`: A list containing the model setup in the same form it was passed.

`obj`: A list returned from `TMB::MakeADFun()` in the same form it was passed.

`opt`: A list containing the optimized model in the same form it was passed.

`max_gradient`: The maximum gradient found when optimizing the model. The default is NA, which means that the model was not optimized.

`report`: A list containing the model report from `obj[["report"]]()`.

`sdreport`: An object with the `sdreport` class containing the output from `TMB::sdreport(obj)`.

`timing`: The length of time it took to run the model if it was optimized.

`version`: The package version of FIMS used to fit the model or at least the version used to create this output, which will not always be the same if you are running this function yourself.

`model_output`: The FIMS model output as a JSON string.

Description

All constructor functions take a single input and build an object specific to the needs of each model type within **FIMS**. FIMSFrame is the parent class. Future, associated child classes will have the additional slots needed for different types of models.

Usage

```
FIMSFrame(data)
```

Arguments

data	A data.frame that contains the necessary columns to construct a FIMSFrame-class object. Currently, those columns are type, name, age, length, timing, value, unit, and uncertainty. See the data_big object in FIMS, e.g., data(data_big, package = "FIMS").
------	--

Details

data:

The input data are both sorted (see the section below on sorting) and expanded before returning them in the data slot.

Ages:

Currently, ages must be integers, i.e., FIMS cannot accommodate numeric ages like age 1.5 but we hope that this is something that we will be able to accommodate in the future. Additionally, the first age in your data set will be the age at which fish recruit to the population. So, if you do not have age-0 age compositions and age two is the first age you have in your composition data, then recruitment will happen at age two. Furthermore, you must have composition entries for every integer between your first and last age in the model. For example, you cannot bin fish into bins that span multiple years, you must have age-2, age-3, and age-4 not just age-2 and age-4 fish in your composition data.

Sorting:

It is important that the order of the rows in the data are correct but it is not expected that the user will do this. Instead, the returned data are sorted using `dplyr::arrange()` before placing them in the data slot. Data are first sorted by data type, placing all weight_at_age data next to other weight_at_age data and all landings data next to landings data. Thus, age_comp data will come first because their type is "age" and "a" is first in the alphabet. All other types will follow according to their order in the alphabet. Next, within each type, data are organized by fleet. So, age_comp information for fleet1 will come before survey1. Next, all data within type and fleet are arranged by timing, e.g., by year. That is the end of the sorting for time series data like landings and indices. The biological data are further sorted by bin. Thus, age_comp information will be arranged as follows:

type	name	timing	age	value
age_comp	fleet1	2022	1	0.3
age_comp	fleet1	2022	2	0.7
age_comp	fleet1	2023	1	0.5

length_comp data are sorted the same way but by length bin instead of by age bin. It becomes more complicated for the age_to_length_conversion data, which are sorted by type, name, timing, age, and then length. So, a full set of length, e.g., length 10, length 20, length 30, etc., is placed together for a given age. After that age, another entire set of length information will be provided for that next age. Once the year is complete for a given fleet then the next year will begin.

Value

An object of the S4 class FIMSFrame class, or one of its child classes, is validated and then returned. All objects will at a minimum have a slot called data to store the input data frame. Additional slots are dependent on the child class. Use `methods::showClass()` to see all available slots.

See Also

`plot,FIMSFrame,missing-method()` for plotting FIMSFrame objects.

fit_fims	<i>Fit a FIMS model (BETA)</i>
----------	--------------------------------

Description

Fit a FIMS model (BETA)

Usage

```
fit_fims(  
  input,  
  get_sd = TRUE,  
  save_sd = TRUE,  
  number_of_loops = 3,  
  optimize = TRUE,  
  number_of_newton_steps = 0,  
  control = list(eval.max = 10000, iter.max = 10000, trace = 0),  
  filename = NULL  
)
```

Arguments

input	Input list as returned by <code>initialize_fims()</code> .
get_sd	A boolean specifying if the <code>TMB::sdreport()</code> should be calculated?
save_sd	A logical, with the default TRUE, indicating whether the sdreport is returned in the output. If FALSE, the slot for the report will be empty.
number_of_loops	A positive integer specifying the number of iterations of the optimizer that will be performed to improve the gradient. The default is three, leading to four total optimization steps.
optimize	Optimize (TRUE, default) or (FALSE) build and return a list containing the obj and report slot.
number_of_newton_steps	The number of Newton steps using the inverse Hessian to do after optimization. Not yet implemented.
control	A list of optimizer settings passed to <code>stats::nlminb()</code> . The the default is a list of length three with <code>eval.max = 1000</code> , <code>iter.max = 10000</code> , and <code>trace = 0</code> .
filename	Character string giving a file name to save the fitted object as an RDS object. Defaults to 'fit.RDS', and a value of NULL indicates not to save it. If specified, it must end in .RDS. The file is written to folder given by <code>input[["path"]]</code> . Not yet implemented.

Details

This function is a beta version still and subject to change without warning.

Value

An object of class FIMSFit is returned, where the structure is the same regardless if `optimize = TRUE` or not. Uncertainty information is only included in the estimates slot if `get_sd = TRUE`.

See Also

- `FIMSFit()`

get_FIMSFit

Get a slot in a FIMSFit object

Description

There is an accessor function for each slot in the S4 class FIMSFit, where the function is named `get_*`() and the star can be replaced with the slot name, e.g., `get_input()`. These accessor functions are the preferred way to access objects stored in the available slots.

Usage

```
get_input(x)

## S4 method for signature 'FIMSFit'
get_input(x)

get_report(x)

## S4 method for signature 'FIMSFit'
get_report(x)

get_obj(x)

## S4 method for signature 'FIMSFit'
get_obj(x)

get_opt(x)

## S4 method for signature 'FIMSFit'
get_opt(x)

get_max_gradient(x)

## S4 method for signature 'FIMSFit'
get_max_gradient(x)

get_sdreport(x)

## S4 method for signature 'FIMSFit'
get_sdreport(x)

get_estimates(x)

## S4 method for signature 'FIMSFit'
get_estimates(x)

get_number_of_parameters(x)

## S4 method for signature 'FIMSFit'
get_number_of_parameters(x)

get_timing(x)

## S4 method for signature 'FIMSFit'
get_timing(x)

get_version(x)
```

```
## S4 method for signature 'FIMSFit'
get_version(x)

get_model_output(x)

## S4 method for signature 'FIMSFit'
get_model_output(x)
```

Arguments

x Output returned from `fit_fims()`.

Value

`get_input()` returns the list that was used to fit the FIMS model, which is the returned object from `create_default_parameters()`.

`get_report()` returns the TMB report, where anything that is flagged as reportable in the C++ code is returned.

`get_obj()` returns the output from `TMB::MakeADFun()`.

`get_opt()` returns the output from `nlminb()`, which is the minimizer used in `fit_fims()`.

`get_max_gradient()` returns the maximum gradient found when optimizing the model.

`get_sdreport()` returns the list from `TMB::sdreport()`.

`get_estimates()` returns a tibble of parameter values and their uncertainties from a fitted model.

`get_number_of_parameters()` returns a vector of integers specifying the number of fixed-effect parameters and the number of random-effect parameters in the model.

`get_timing()` returns the amount of time it took to run the model in seconds as a `difftime` object.

`get_version()` returns the `package_version` of FIMS that was used to fit the model.

`get_model_output()` returns the finalized FIMS output as a JSON list.

See Also

- `fit_fims()`
- `create_default_parameters()`

get_FIMSFrame

Get a slot in a FIMSFrame object

Description

There is an accessor function for each slot in the S4 class `FIMSFrame`, where the function is named `get_*`() and the star can be replaced with the slot name, e.g., `get_data()`. These accessor functions are the preferred way to access objects stored in the available slots.

Usage

```
get_data(x)

## S4 method for signature 'FIMSFrame'
get_data(x)

## S4 method for signature 'data.frame'
get_data(x)

get_fleets(x)

## S4 method for signature 'FIMSFrame'
get_fleets(x)

## S4 method for signature 'data.frame'
get_fleets(x)

get_n_fleets(x)

## S4 method for signature 'FIMSFrame'
get_n_fleets(x)

## S4 method for signature 'data.frame'
get_n_fleets(x)

get_n_years(x)

## S4 method for signature 'FIMSFrame'
get_n_years(x)

## S4 method for signature 'data.frame'
get_n_years(x)

get_start_year(x)

## S4 method for signature 'FIMSFrame'
get_start_year(x)

## S4 method for signature 'data.frame'
get_start_year(x)

get_end_year(x)

## S4 method for signature 'FIMSFrame'
get_end_year(x)

## S4 method for signature 'data.frame'
get_end_year(x)
```

```
get_ages(x)

## S4 method for signature 'FIMSFrame'
get_ages(x)

## S4 method for signature 'data.frame'
get_ages(x)

get_n_ages(x)

## S4 method for signature 'FIMSFrame'
get_n_ages(x)

## S4 method for signature 'data.frame'
get_n_ages(x)

get_lengths(x)

## S4 method for signature 'FIMSFrame'
get_lengths(x)

## S4 method for signature 'data.frame'
get_lengths(x)

get_n_lengths(x)

## S4 method for signature 'FIMSFrame'
get_n_lengths(x)

## S4 method for signature 'data.frame'
get_n_lengths(x)
```

Arguments

x An object returned from `FIMSFrame()`.

Value

`get_data()` returns a data frame of the class `tbl_df` containing data for a FIMS model in a long format. The tibble will potentially have the following columns depending if it fits to ages and lengths or just one of them: type, name, age, length, timing, value, unit, and uncertainty.

`get_fleets()` returns a vector of strings containing the fleet names.

`get_n_fleets()` returns an integer specifying the number of fleets in the model, where fleets is inclusive of both fishing fleets and survey vessels.

`get_n_years()` returns an integer specifying the number of years in the model.

`get_start_year()` returns an integer specifying the start year of the model.

`get_end_year()` returns an integer specifying the end year of the model.

`get_ages()` returns a vector of age bins used in the model.

`get_n_ages()` returns an integer specifying the number of age bins used in the model.

`get_lengths()` returns a vector of length bins used in the model.

`get_n_lengths()` returns an integer specifying the number of length bins used in the model.

```
initialize_data_distribution
```

Set up a new distribution for a data type or a process

Description

Use `methods::new()` to set up a distribution within an existing module with the necessary linkages between the two. For example, a fleet module will need a distributional assumption for parts of the data associated with it, which requires the use of `initialize_data_distribution()`, and a recruitment module, like the Beverton–Holt stock–recruit relationship, will need a distribution associated with the recruitment deviations, which requires `initialize_process_distribution()`.

Usage

```
initialize_data_distribution(
  module,
  family = NULL,
  sd = tibble::tibble(value = 1, estimation_type = "constant"),
  data_type = c("landings", "index", "agecomp", "lengthcomp")
)

initialize_process_distribution(
  module,
  par,
  family = NULL,
  sd = tibble::tibble(value = 1, estimation_type = "fixed_effects")
)

initialize_process_structure(module, par)
```

Arguments

<code>module</code>	An identifier to a C++ fleet module that is linked to the data of interest.
<code>family</code>	A description of the error distribution and link function to be used in the model. The argument takes a family class, e.g., <code>stats::gaussian(link = "identity")</code> .
<code>sd</code>	A list of length two. The first entry is named "value" and it stores the initial values (scalar or vector) for the relevant standard deviations. The default is value = 1. The second entry is named "estimation_type" and it stores a vector of

	booleans (default = "constant") is a string indicating whether or not standard deviation is estimated as a fixed effect or held constant. If "value" is a vector and "estimation_type" is a scalar, the single value specified "estimation_type" value will be repeated to match the length of value. Otherwise, the dimensions of the two must match.
data_type	A string specifying the type of data that the distribution will be fit to. Allowable types include "landings", "index", "agecomp", and "lengthcomp" and the default is landings.
par	A string specifying the parameter name the distribution applies to. Parameters must be members of the specified module. Use <code>methods::show(module)</code> to obtain names of parameters within the module.

Value

A reference class. is returned. Use `methods::show()` to view the various Rcpp class fields, methods, and documentation.

Examples

```
## Not run:
# Set up a new data distribution
n_years <- 30
# Create a new fleet module
fleet <- methods::new(Fleet)
# Create a distribution for the fleet module
fleet_distribution <- initialize_data_distribution(
  module = fishing_fleet,
  family = lognormal(link = "log"),
  sd = list(
    value = rep(sqrt(log(0.01^2 + 1)), n_years),
    estimation_type = rep("constant", n_years) # Can be a single "constant"
  ),
  data_type = "index"
)

# Set up a new process distribution
# Create a new recruitment module
recruitment <- methods::new(BevertonHoltRecruitment)
# view parameter names of the recruitment module
methods::show(BevertonHoltRecruitment)
# Create a distribution for the recruitment module
recruitment_distribution <- initialize_process_distribution(
  module = recruitment,
  par = "log_devs",
  family = gaussian(),
  sd = list(value = 0.4, estimation_type = "constant")
)

## End(Not run)
```

initialize_fims	<i>Initialize C++ modules via Rcpp for a FIMS model</i>
-----------------	---

Description

This function uses information from a parameter data frame that stores the model specifications and a `FIMSFrame` object that stores the data to instantiate, i.e., create an instance of a class, the required C++ modules. Several C++ modules are needed to run a FIMS model and the required modules will be different for each model type. For example, for a catch-at-age model one needs to instantiate recruitment, growth, and maturity modules and at least one fleet and population module.

Usage

```
initialize_fims(parameters, data)
```

Arguments

parameters	A tibble returned from <code>create_default_parameters()</code> . The tibble can be nested, i.e., contain a data column, or unnested, i.e., <code>tidyr::unnest(create_default_parameters(), cols = "data")</code> . Regardless, it is the primary source of information for what is initialized. That is, if a fleet exists in the data but parameter information for how to specify selectivity for that fleet is not provided, then selectivity will not be initialized for that fleet.
data	An S4 object with the <code>FIMSFrame</code> class, which is returned from <code>FIMSFrame()</code> . Passing the data is required because initialization of the modules requires passing the data and information regarding the uncertainty of that data, i.e., input sample sizes for the multinomial distribution.

Value

A list is returned with two elements, `parameters` and `model`. The list can be passed to the input argument of `fit_fims()` to fit the model. The first element of the list can also be passed to the `parameters` argument of `TMB::MakeADFun()` if you wish to have more control over the model-fitting process. The `model` element of the returned list stores the instantiated C++ model module, e.g., the results of `methods::new(CatchAtAge)` for a catch-at-age model. It is important that you only have one FIMS model initialized in your R workspace at a time. Thus, after you initialize and fit the model, you should run `clear()`.

See Also

- `create_default_configurations()`
- `create_default_parameters()`
- `FIMSFrame()`
- `fit_fims()`
- `clear()`

Examples

```
## Not run:
# Prepare data for FIMS model
data("data_big", package = "FIMS")
data_4_model <- FIMSFrame(data_big)
# Instantiate modules
parameters_list <- data_4_model |>
  create_default_configurations() |>
  create_default_parameters(data = data_4_model) |>
  initialize_fims(data = data_4_model)
clear()

## End(Not run)
```

is.FIMSFit	<i>Check if an object is of class FIMSFit</i>
------------	---

Description

Check if an object is of class FIMSFit

Usage

```
is.FIMSFit(x)
```

Arguments

x	Returned list from <code>fit_fims()</code> .
---	--

is_fims_verbose	<i>Should FIMS be verbose?</i>
-----------------	--------------------------------

Description

Verbosity is set globally for FIMS using `options(rlib_message_verbosity = "quiet")` to stop the printing of messages from `cli::cli_inform()`. Using a global option allows for verbose to not have to be an argument to every function. All `cli::cli_abort()` messages are printed to the console no matter what the global option is set to.

Usage

```
is_fims_verbose()
```

Value

A logical is returned where TRUE ensures messages from `cli::cli_inform()` are printed to the console.

Examples

```
# function is not exported
## Not run:
FIMS:::is_fims_verbose()

## End(Not run)
```

lognormal

*Distributions not available in the stats package***Description**

Family objects provide a convenient way to specify the details of the models used by functions such as `stats::glm()`. These functions within this package are not available within the stats package but are designed in a similar manner.

Usage

```
lognormal(link = "log")

multinomial(link = "logit")
```

Arguments

link A string specifying the model link function. For example, "identity" or "log" are appropriate names for the `stats::gaussian()` distribution. "log" and "logit" are the defaults for the lognormal and the multinomial, respectively.

Value

An object of class family (which has a concise print method). This particular family has a truncated length compared to other distributions in `stats::family()`.

family character: the family name.
link character: the link name.

See Also

- `stats::family()`
- `stats::gaussian()`
- `stats::glm()`
- `stats::power()`
- `stats::make.link()`

Examples

```
a_family <- multinomial()
a_family[["family"]]
a_family[["link"]]
```

model_	<i>Get a vector of data to be passed to a FIMS module from a FIMSFrame object</i>
--------	---

Description

There is an accessor function for each data type needed to run a FIMS model. A FIMS model accepts vectors of data and thus each of the `model_*()` functions, where the star can be replaced with the data type separated by underscores, e.g., `weight_at_age`. These accessor functions are the preferred way to pass data to a FIMS module because the data will have the appropriate indexing.

Usage

```
model_landings(x, fleet_name)

## S4 method for signature 'FIMSFrame'
model_landings(x, fleet_name)

## S4 method for signature 'data.frame'
model_landings(x, fleet_name)

model_index(x, fleet_name)

## S4 method for signature 'FIMSFrame'
model_index(x, fleet_name)

## S4 method for signature 'data.frame'
model_index(x, fleet_name)

model_age_comp(x, fleet_name)

## S4 method for signature 'FIMSFrame'
model_age_comp(x, fleet_name)

## S4 method for signature 'data.frame'
model_age_comp(x, fleet_name)

model_length_comp(x, fleet_name)

## S4 method for signature 'FIMSFrame'
model_length_comp(x, fleet_name)

## S4 method for signature 'data.frame'
model_length_comp(x, fleet_name)

model_weight_at_age(x)
```

```
## S4 method for signature 'FIMSFrame'
model_weight_at_age(x)

## S4 method for signature 'data.frame'
model_weight_at_age(x)

model_age_to_length_conversion(x)

## S4 method for signature 'FIMSFrame'
model_age_to_length_conversion(x)

## S4 method for signature 'data.frame'
model_age_to_length_conversion(x)
```

Arguments

x	An object returned from FIMSFrame() .
fleet_name	A string, or vector of strings, specifying the name of the fleet(s) of interest that you want landings data for. The strings must exactly match strings in the column "name" of <code>get_data(x)</code> .

Details

Age_to_length_conversion data, i.e., the proportion of age "a" that are length "l", are used to convert lengths (input data) to ages (modeled) as a way to fit length data without estimating growth.

`model_landings()`:

Returns a numeric vector of landings data (type "landings") for the specified fleet(s).

`model_index()`:

Returns a numeric vector of index data (type "index") for the specified fleet(s).

`model_age_comp()`:

Returns a numeric vector of age-composition data (type "age_comp") for the specified fleet(s).

`model_length_comp()`:

Returns a numeric vector of length-composition data (type "length_comp") for the specified fleet(s). This accessor requires that age-to-length conversion data are present in the `FIMSFrame` object.

`model_weight_at_age()`:

Returns a numeric vector of weight-at-age values (type "weight_at_age"), ordered by age and year (plus one additional year) and using -999 to indicate missing values. When multiple fleets are present, values are averaged across fleets.

`model_age_to_length_conversion()`:

Returns a numeric vector of age-to-length conversion values. Values are ordered by age and length in the order defined by [FIMSFrame\(\)](#). When multiple values are present across a single age and length, they are averaged because age_to_length_conversion data cannot vary across fleets or time.

Value

All of the `model_*()` functions return vectors of data. Currently, the order of the data is the same order as the data frame because no arranging is done in `FIMSFrame()` and the function just extracts the appropriate column.

```
plot,FIMSFrame,missing-method
```

Plot a FIMSFrame object

Description

Use `ggplot2::geom_point()` to plot summary information stored in the data slot of the `FIMSFrame` class.

Usage

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

Arguments

<code>x</code>	A <code>FIMSFrame</code> object.
<code>y</code>	Unused (inherited from R base).
<code>...</code>	Unused (inherited from R base).

Value

A `ggplot2` object is returned that uses `stockplotr::theme_noaa()`. There will be one panel per input type with fleet-specific information denoted using colors.

Examples

```
## Not run:
data("data_big", package = "FIMS")
data_4_model <- FIMSFrame(data_big)
plot(data_4_model)

## End(Not run)
```

remove_test_data	<i>Remove test data</i>
------------------	-------------------------

Description

Intended for developers to remove test data to run the tests from fresh. Developers should run this function before testing if changes affect FIMS input or output.

Usage

```
remove_test_data()
```

Examples

```
## Not run:  
remove_test_data()  
  
## End(Not run)
```

reshape_json_estimates	<i>Reshape JSON estimates</i>
------------------------	-------------------------------

Description

This function processes the finalized FIMS JSON output and reshapes the parameter estimates into a structured tibble for easier analysis and manipulation.

Usage

```
reshape_json_estimates(model_output)
```

Arguments

model_output	A JSON object containing the finalized FIMS output as returned from <code>get_output()</code> , which is an internal function to each model family.
--------------	---

Value

A tibble containing the reshaped parameter estimates.

 reshape_tmb_estimates *Reshape TMB estimates*

Description

This function processes the TMB std and reshapes them into a structured tibble for easier analysis and manipulation.

Usage

```
reshape_tmb_estimates(obj, sdreport = NULL, opt = NULL, parameter_names)
```

Arguments

obj	An object returned from <code>TMB::MakeADFun()</code> .
sdreport	An object of the sdreport class as returned from <code>TMB::sdreport()</code> .
opt	An object returned from an optimizer, typically from <code>stats::nlminb()</code> , used to fit a TMB model. If the model is not optimized, opt is an empty list and is not used in the function.
parameter_names	A character vector of parameter names. This is used to identify the parameters in the std object.

Value

A tibble containing the reshaped estimates (i.e., parameters and derived quantities).

 run_gtest *Run the google test suite*

Description

Intended for developers to run the google test suite from R.

Usage

```
run_gtest(...)
```

Arguments

...	Additional arguments to ctest <code>--test-dir build</code> such as <code>"--rerun-failed --output-on-failure"</code> or <code>--parallel 16</code> if you want it to run on multiple cores.
-----	--

Details

The output from running the tests should look something like the following:

```
Internal ctest changing into directory: C:/github/NOAA-FIMS/FIMS/build
Test project C:/github_repos/NOAA-FIMS_org/FIMS/build
  Start 1: dlognorm.use_double_inputs
1/5 Test #1: dlognorm.use_double_inputs ..... Passed    0.04 sec
  Start 2: dlognorm.use_int_inputs
2/5 Test #2: dlognorm.use_int_inputs ..... Passed    0.04 sec
  Start 3: modelTest.eta
3/5 Test #3: modelTest.eta ..... Passed    0.04 sec
  Start 4: modelTest.nll
4/5 Test #4: modelTest.nll ..... Passed    0.04 sec
  Start 5: modelTest.evaluate
5/5 Test #5: modelTest.evaluate ..... Passed    0.04 sec

100% tests passed, 0 tests failed out of 5
```

If a test fails, the output will look something like the following:

```
Internal ctest changing into directory: C:/github/NOAA-FIMS/FIMS/build
Test project C:/github/NOAA-FIMS/FIMS/build
  Start 1: dlognorm.use_double_inputs
1/7 Test #1: dlognorm.use_double_inputs ..... Passed    0.04 sec
  Start 2: dlognorm.use_int_inputs
2/7 Test #2: dlognorm.use_int_inputs ..... Passed    0.04 sec
  Start 3: modelTest.eta
3/7 Test #3: modelTest.eta ..... Passed    0.04 sec
  Start 4: modelTest.nll
4/7 Test #4: modelTest.nll ..... Passed    0.04 sec
  Start 5: modelTest.evaluate
5/7 Test #5: modelTest.evaluate ..... Passed    0.04 sec
  Start 6: dlognormTest.DoubleInput
6/7 Test #6: dlognormTest.DoubleInput ..... Passed    0.04 sec
  Start 7: dlognormTest.IntInput
7/7 Test #7: dlognormTest.IntInput .....***Failed    0.04 sec

86% tests passed, 1 tests failed out of 7
```

Total Test time (real) = 0.28 sec

The following tests FAILED:

7 - dlognormTest.IntInput (Failed)

Errors while running CTest

Output from these tests are in: C:/github/NOAA-FIMS/FIMS/build/Testing/Temporary/LastTest.log
Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.

`run_r_integration_tests`*Run R integration tests*

Description

Intended for developers to run the R integration tests.

Usage

```
run_r_integration_tests()
```

Examples

```
## Not run:  
run_r_integration_tests()  
  
## End(Not run)
```

`run_r_unit_tests`*Run R unit tests*

Description

Intended for developers to run the R unit tests.

Usage

```
run_r_unit_tests()
```

Examples

```
## Not run:  
run_r_unit_tests()  
  
## End(Not run)
```

setup_and_run_gtest	<i>Setup and run the google test suite</i>
---------------------	--

Description

Intended for developers to set up their local environment and run the google test suite from R.

Usage

```
setup_and_run_gtest(...)
```

Arguments

...	Additional arguments to ctest --test-dir build such as "--rerun-failed --output-on-failure" or --parallel 16 if you want it to run on multiple cores.
-----	---

Examples

```
## Not run:  
setup_and_run_gtest("--rerun-failed --output-on-failure")  
  
## End(Not run)
```

setup_gtest	<i>Set up your local environment to run the google tests locally</i>
-------------	--

Description

Intended for developers to set up their local environment prior to running the integration tests.

Usage

```
setup_gtest()
```

Examples

```
## Not run:  
setup_gtest()  
  
## End(Not run)
```

use_gtest_template	<i>Create tests/gtest/test_*.cpp test file and register it in CMakeLists.txt</i>
--------------------	--

Description

This helper function generates a GoogleTest (gtest) C++ template file for a given function and appends lines to CMakeLists.txt to register the test.

Usage

```
use_gtest_template(name = "FileName_ClassName_FunctionName")
```

Arguments

name	A string representing the combined name for the C++ test file and CMake executable target. It must follow the format FileName_ClassName_FunctionName, where • FileName is the C++ source file name (e.g., "Logistic") • ClassName is the C++ class name (e.g., "LogisticSelectivity") • FunctionName is the C++ function name (e.g., "Evaluate") and use under-scores to separate each component. For example: Logistic_LogisticSelectivity_Evaluate. If the function is not a member of a class, use a placeholder for ClassName. For example: FIMSMath_ClassName_Logistic. The default is FileName_ClassName_FunctionName.
------	---

Value

If successful, this function invisibly returns TRUE to allow for the chaining of commands. If the function is unsuccessful, an error message is returned.

Three messages are also returned from the usethis package, which is used by this function. The first states where the FIMS project is on your computer. The second states the file path of the newly created file. The file will not be automatically opened. The third states the test has been registered in tests/gtest/CMakeLists.txt.

Examples

```
## Not run:
# Create a new test file named
# "test_Logistic_LogisticSelectivity_Evaluate.cpp" for
# `LogisticSelectivity::evaluate()` in
# `inst/include/population_dynamics/selectivity/functors/logistic.hpp`.

FIMS:::use_gtest_template(name = "Logistic_LogisticSelectivity_Evaluate")

## End(Not run)
```

use_testthat_template *Create tests/testthat/test-*.R test file*

Description

This helper function creates a new test file for testthat using a template available within the templates folder of this package. The test file is created under the "tests/testthat" directory with a name based on the input argument name. If the test file already exists, an error message is returned and no changes are made.

Usage

```
use_testthat_template(name)
```

Arguments

name	A character string providing the name of the R function that you want to test. The name will be used to create the file name, i.e., tests/testthat/test-{name}.R. If name is not specified, the function will not be able to create a file name and an error will be returned.
------	--

Details

There are three minimum testing criteria for FIMS, which should be validated for every R function within the package. The template file sets up a section for each of the three following test criteria:

1. Input and output correctness (IO correctness): ensure that the function behaves as expected with correct inputs and returns the expected outputs.
2. Edge-case handling (Edge handling): validate the function's performance with invalid inputs and unusual scenarios.
3. Built-in errors and warnings (Error handling): confirm that appropriate error and warning messages are triggered under exceptional conditions.

Above every expectation within the test file there should be a call to #' @description that fits on one line describing the test. The information will be used in the bookdown report of the testing results.

Value

If successful, this function invisibly returns TRUE to allow for the chaining of commands. If the function is unsuccessful, an error message is returned.

Two messages are also returned from the usethis package, which is used by this function. The first states where the FIMS project is on your computer. The second states the file path of the newly created file. The file will not be automatically opened.

Examples

```
# Create a new test file named "test-new_function" for `new_function()`  
## Not run:  
FIMS:::use_testthat_template("new_function")  
  
## End(Not run)
```

Index

- * **FIMSFrame**
 - FIMSFrame, [12](#)
 - get_FIMSFrame, [16](#)
 - model_, [24](#)
- * **datasets**
 - data_big, [9](#)
 - fims_input_types, [10](#)
- * **developer**
 - remove_test_data, [27](#)
 - run_gtest, [28](#)
 - run_r_integration_tests, [30](#)
 - run_r_unit_tests, [30](#)
 - setup_and_run_gtest, [31](#)
 - setup_gtest, [31](#)
 - use_gtest_template, [32](#)
 - use_testthat_template, [33](#)
- * **distribution**
 - initialize_data_distribution, [19](#)
 - lognormal, [23](#)
- * **fit_fims**
 - FIMSFit, [10](#)
 - fit_fims, [13](#)
 - get_FIMSFit, [14](#)
 - is.FIMSFit, [22](#)
- AgeComp (Cpp_classes), [3](#)
- BevertonHoltRecruitment (Cpp_classes), [3](#)
- CatchAtAge (Cpp_classes), [3](#)
- clear (Cpp_functions), [4](#)
- clear(), [21](#)
- Cpp_classes, [3](#)
- Cpp_functions, [4](#)
- create_default_configurations, [5](#)
- create_default_configurations(), [7](#), [8](#), [21](#)
- create_default_parameters, [7](#)
- create_default_parameters(), [5](#), [6](#), [16](#), [21](#)
- CreateTMBModel (Cpp_functions), [4](#)
- data_big, [9](#)
- DlnormDistribution (Cpp_classes), [3](#)
- DmultinomDistribution (Cpp_classes), [3](#)
- DnormDistribution (Cpp_classes), [3](#)
- DoubleLogisticSelectivity (Cpp_classes), [3](#)
- dplyr::arrange(), [12](#)
- EWAAGrowth (Cpp_classes), [3](#)
- fims_input_types, [10](#)
- FIMSFit, [10](#)
- FIMSFit(), [14](#)
- FIMSFrame, [12](#)
- FIMSFrame(), [5–8](#), [18](#), [21](#), [25](#), [26](#)
- fit_fims, [13](#)
- fit_fims(), [10](#), [11](#), [16](#), [21](#), [22](#)
- Fleet (Cpp_classes), [3](#)
- get_ages (get_FIMSFrame), [16](#)
- get_ages(), [19](#)
- get_ages, data.frame-method (get_FIMSFrame), [16](#)
- get_ages, FIMSFrame-method (get_FIMSFrame), [16](#)
- get_data (get_FIMSFrame), [16](#)
- get_data(), [16](#), [18](#)
- get_data, data.frame-method (get_FIMSFrame), [16](#)
- get_data, FIMSFrame-method (get_FIMSFrame), [16](#)
- get_end_year (get_FIMSFrame), [16](#)
- get_end_year(), [19](#)
- get_end_year, data.frame-method (get_FIMSFrame), [16](#)
- get_end_year, FIMSFrame-method (get_FIMSFrame), [16](#)
- get_estimates (get_FIMSFit), [14](#)
- get_estimates(), [16](#)

get_estimates, FIMSFit-method
 (get_FIMSFit), 14
 get_FIMSFit, 14
 get_FIMSFrame, 16
 get_fixed (Cpp_functions), 4
 get_fleets (get_FIMSFrame), 16
 get_fleets(), 18
 get_fleets, data.frame-method
 (get_FIMSFrame), 16
 get_fleets, FIMSFrame-method
 (get_FIMSFrame), 16
 get_input (get_FIMSFit), 14
 get_input(), 14, 16
 get_input, FIMSFit-method (get_FIMSFit),
 14
 get_lengths (get_FIMSFrame), 16
 get_lengths(), 19
 get_lengths, data.frame-method
 (get_FIMSFrame), 16
 get_lengths, FIMSFrame-method
 (get_FIMSFrame), 16
 get_log (Cpp_functions), 4
 get_log_errors (Cpp_functions), 4
 get_log_warnings (Cpp_functions), 4
 get_max_gradient (get_FIMSFit), 14
 get_max_gradient(), 16
 get_max_gradient, FIMSFit-method
 (get_FIMSFit), 14
 get_model_output (get_FIMSFit), 14
 get_model_output(), 16
 get_model_output, FIMSFit-method
 (get_FIMSFit), 14
 get_n_ages (get_FIMSFrame), 16
 get_n_ages(), 19
 get_n_ages, data.frame-method
 (get_FIMSFrame), 16
 get_n_ages, FIMSFrame-method
 (get_FIMSFrame), 16
 get_n_fleets (get_FIMSFrame), 16
 get_n_fleets(), 18
 get_n_fleets, data.frame-method
 (get_FIMSFrame), 16
 get_n_fleets, FIMSFrame-method
 (get_FIMSFrame), 16
 get_n_lengths (get_FIMSFrame), 16
 get_n_lengths(), 19
 get_n_lengths, data.frame-method
 (get_FIMSFrame), 16
 get_n_lengths, FIMSFrame-method
 (get_FIMSFrame), 16
 get_n_years (get_FIMSFrame), 16
 get_n_years(), 18
 get_n_years, data.frame-method
 (get_FIMSFrame), 16
 get_n_years, FIMSFrame-method
 (get_FIMSFrame), 16
 get_number_of_parameters (get_FIMSFit),
 14
 get_number_of_parameters(), 16
 get_number_of_parameters, FIMSFit-method
 (get_FIMSFit), 14
 get_obj (get_FIMSFit), 14
 get_obj(), 16
 get_obj, FIMSFit-method (get_FIMSFit), 14
 get_opt (get_FIMSFit), 14
 get_opt(), 16
 get_opt, FIMSFit-method (get_FIMSFit), 14
 get_parameter_names (Cpp_functions), 4
 get_random (Cpp_functions), 4
 get_random_names (Cpp_functions), 4
 get_report (get_FIMSFit), 14
 get_report(), 16
 get_report, FIMSFit-method
 (get_FIMSFit), 14
 get_sdreport (get_FIMSFit), 14
 get_sdreport(), 16
 get_sdreport, FIMSFit-method
 (get_FIMSFit), 14
 get_start_year (get_FIMSFrame), 16
 get_start_year(), 18
 get_start_year, data.frame-method
 (get_FIMSFrame), 16
 get_start_year, FIMSFrame-method
 (get_FIMSFrame), 16
 get_timing (get_FIMSFit), 14
 get_timing(), 16
 get_timing, FIMSFit-method
 (get_FIMSFit), 14
 get_version (get_FIMSFit), 14
 get_version(), 16
 get_version, FIMSFit-method
 (get_FIMSFit), 14
 Index (Cpp_classes), 3
 initialize_data_distribution, 19
 initialize_fims, 21
 initialize_fims(), 11, 14

[initialize_process_distribution](#)
 ([initialize_data_distribution](#)),
 19
[initialize_process_structure](#)
 ([initialize_data_distribution](#)),
 19
[inv_logit](#) ([Cpp_functions](#)), 4
[is.FIMSFit](#), 22
[is_fims_verbose](#), 22

[Landings](#) ([Cpp_classes](#)), 3
[LengthComp](#) ([Cpp_classes](#)), 3
[log_error](#) ([Cpp_functions](#)), 4
[log_info](#) ([Cpp_functions](#)), 4
[log_warning](#) ([Cpp_functions](#)), 4
[LogDevsRecruitmentProcess](#)
 ([Cpp_classes](#)), 3
[LogisticMaturity](#) ([Cpp_classes](#)), 3
[LogisticSelectivity](#) ([Cpp_classes](#)), 3
[logit](#) ([Cpp_functions](#)), 4
[lognormal](#), 23
[LogRRRecruitmentProcess](#) ([Cpp_classes](#)), 3

[methods::new\(\)](#), 19
[methods::show\(\)](#), 20
[methods::showClass\(\)](#), 13
[model_](#), 24
[model_age_comp](#) ([model_](#)), 24
[model_age_comp](#), [data.frame](#)-method
 ([model_](#)), 24
[model_age_comp](#), [FIMSFrame](#)-method
 ([model_](#)), 24
[model_age_to_length_conversion](#)
 ([model_](#)), 24
[model_age_to_length_conversion](#), [data.frame](#)-method
 ([model_](#)), 24
[model_age_to_length_conversion](#), [FIMSFrame](#)-method
 ([model_](#)), 24
[model_index](#) ([model_](#)), 24
[model_index](#), [data.frame](#)-method ([model_](#)),
 24
[model_index](#), [FIMSFrame](#)-method ([model_](#)),
 24
[model_landings](#) ([model_](#)), 24
[model_landings](#), [data.frame](#)-method
 ([model_](#)), 24
[model_landings](#), [FIMSFrame](#)-method
 ([model_](#)), 24
[model_length_comp](#) ([model_](#)), 24

[model_length_comp](#), [data.frame](#)-method
 ([model_](#)), 24
[model_length_comp](#), [FIMSFrame](#)-method
 ([model_](#)), 24
[model_weight_at_age](#) ([model_](#)), 24
[model_weight_at_age](#), [data.frame](#)-method
 ([model_](#)), 24
[model_weight_at_age](#), [FIMSFrame](#)-method
 ([model_](#)), 24
[multinomial](#) ([lognormal](#)), 23

[nlminb\(\)](#), 16

[Parameter](#) ([Cpp_classes](#)), 3
[ParameterVector](#) ([Cpp_classes](#)), 3
[plot](#), [FIMSFrame](#), [missing](#)-method, 26
[Population](#) ([Cpp_classes](#)), 3

[RealVector](#) ([Cpp_classes](#)), 3
[remove_test_data](#), 27
[reshape_json_estimates](#), 27
[reshape_tmb_estimates](#), 28
[run_gtest](#), 28
[run_r_integration_tests](#), 30
[run_r_unit_tests](#), 30

[set_fixed](#) ([Cpp_functions](#)), 4
[set_log_throw_on_error](#) ([Cpp_functions](#)),
 4
[set_random](#) ([Cpp_functions](#)), 4
[setup_and_run_gtest](#), 31
[setup_gtest](#), 31
[SharedInt](#) ([Cpp_classes](#)), 3
[SharedReal](#) ([Cpp_classes](#)), 3
[SharedString](#) ([Cpp_classes](#)), 3
[stats::family\(\)](#), 23
[stats::gaussian\(\)](#), 23
[stats::glm\(\)](#), 23
[stats::make.link\(\)](#), 23
[stats::nlminb\(\)](#), 11, 14, 28
[stats::power\(\)](#), 23
[stockplotr::theme_noaa\(\)](#), 26
[Sys.time\(\)](#), 11

[tidyr::unnest\(\)](#), 5
[TMB::MakeADFun\(\)](#), 11, 16, 21, 28
[TMB::sdreport\(\)](#), 11, 14, 16, 28

[use_gtest_template](#), 32
[use_testthat_template](#), 33