

Package: salmoRodeo (via r-universe)

October 13, 2026

Title A 1-layer ecological lake model in Petersen-Guger matrix notation

Version 0.2

Description The package implements a 1-layer version of the ecological lake model SALMO using a tabular Petersen-Guger matrix description. It simulates main components of the pelagic foodweb in a shallow, non-stratified lake.

Depends R ($\geq 4.1.0$), rodeo

Imports stringr, stats

Suggests quarto, deSolve, knitr

License GPL (≥ 2)

VignetteBuilder knitr, quarto

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL <https://tpetzoldt.github.io/salmoRodeo/>

Config/pak/sysreqs libicu-dev

Repository <https://tpetzoldt.r-universe.dev>

Date/Publication 2025-09-26 13:20:25 UTC

RemoteUrl <https://github.com/tpetzoldt/salmoRodeo>

RemoteRef HEAD

RemoteSha 6d29d2daaca66001df3a10ac6a745669ecc81590

Contents

generate_forcings	2
model_setup	3
replace_abbreviations	5
replace_values	5
set_initval	6
set_pars	7
set_pars2	7

Index**9**

generate_forcings	<i>Generate forcing function specifications for a model.</i>
-------------------	--

Description

This function creates a data frame suitable for defining forcing functions in a rodeo model. It specifies the names of the forcing functions, the corresponding column names in the input data file, and other metadata.

Usage

```
generate_forcings(datafile, columns)
```

Arguments

datafile	The name of the data file that contains the forcing data.
columns	A character vector specifying the names of the columns in the datafile that correspond to the forcing functions. These names will also be used as the function names.

Value

A data frame with columns name, column, file, mode, and default, structured for `rodeo::forcingFunctions`.

Examples

```
# Assuming `rodeo::forcingFunctions` is available and correctly setup
# Example: Create a dummy data file
cat("stagnation\tVE\n",
    "1\t10\n",
    "2\t20\n",
    file = "input_data.tsv")

# Generate forcing function specifications
forcings_spec <- generate_forcings(
  datafile = "input_data.tsv",
  columns = c("stagnation", "VE")
)
print(forcings_spec)
# Note: The actual `forcingFunctions` call within the function
# will generate Fortran/C code based on this data frame.
# For this example, we just show the data frame output.

# Clean up dummy file
unlink("input_data.tsv")
```

model_setup

*Set up and compile a rodeo model.***Description**

This function orchestrates the setup of a rodeo model by reading various definition files (variables, parameters, functions, processes, stoichiometry), applying replacements to process expressions, generating forcing functions, and finally compiling the model.

Usage

```
model_setup(
  base_path = NULL,
  vars = "vars.tsv",
  pars = "pars.tsv",
  funs = "funs.tsv",
  pros = "pros.tsv",
  stoi = "stoi.tsv",
  repl = "replacements.tsv",
  datafile = paste0(system.file("models", package = "salmoRodeo"), "/inputs.tsv"),
  input_columns = c("stagnation", "VE", "ZMIX", "QIN", "IIN", "TE", "PIN", "NIN",
    "POMIN"),
  sources = "functions_extended.f95",
  phyto_groups = 3,
  lib = NULL
)
```

Arguments

base_path	Path to the model files. Can be empty, if vars, pars, funs, pros, soi and repl contain full path and file names.
vars	File name of the variables definition TSV file.
pars	File name of the parameters definition TSV file.
funs	File name of the functions definition TSV file.
pros	File name of the processes definition TSV file.
stoi	File name of the stoichiometry definition TSV file.
repl	File name of the replacements definition TSV file.
datafile	Path and file name of the input data TSV file used for forcings.
input_columns	A character vector of column names from datafile to be used as input forcings.
sources	File name to additional Fortran source files to be compiled with the model (e.g., functions_extended.f95).
phyto_groups	An integer (2 or 3) indicating the number of phytoplankton groups, affecting how process and replacement files are read.
lib	File path to be used for the generated library (without the platform specific extension). Note that any uppercase characters will be converted to lowercase. By default, the file is created in R's temporary folder under a random name.

Value

A rodeo model object, compiled and ready for simulation.

Examples

```
## Not run:
# This example requires dummy files and the 'rodeo' package
# and is skipped in automatic checks.
# Create dummy files
dir.create("model_files", showWarnings = FALSE)
cat("name\tmin\tmax\tunit\tdescription\n",
    "N\t0\tInf\tmmol/m3\tNitrogen\n", file = "model_files/vars.tsv")
cat("name\tdefault\tunit\tdescription\n",
    "KN\t0.1\tmmol/m3\tHalf-saturation constant for N\n", file = "model_files/pars.tsv")
cat("name\texpression\n",
    "f_N\tN/(KN+N)\n", file = "model_files/funs.tsv")
cat("name\texpression\ttwoPhytoGroups\tthreePhytoGroups\n",
    "uptake\tN*f_N\tN*f_N_2\tN*f_N_3\n", file = "model_files/pros.tsv")
cat("process\tN\n",
    "uptake\t-1\n", file = "model_files/stoi.tsv")
cat("pattern\treplacement\ttwoPhytoGroups\tthreePhytoGroups\n",
    "N\tN03\tN_two\tN_three\n", file = "model_files/replacements.tsv")
cat("stagnation\tVE\n",
    "1\t10\n",
    "2\t20\n",
    file = "model_files/inputs.tsv")

# Define a dummy function_extended.f95 (can be empty for this example)
cat("SUBROUTINE dummy_ext()\nEND SUBROUTINE\n", file = "model_files/functions_extended.f95")

# Run model setup
model_obj <- model_setup(
  vars = "model_files/vars.tsv",
  pars = "model_files/pars.tsv",
  funs = "model_files/funs.tsv",
  pros = "model_files/pros.tsv",
  stoi = "model_files/stoi.tsv",
  repl = "model_files/replacements.tsv",
  datafile = "model_files/inputs.tsv",
  input_columns = c("stagnation", "VE"),
  sources = "model_files/functions_extended.f95",
  phyto_groups = 3
)

# Check if model is compiled (it should be)
print(model_obj$compiled())

# Clean up dummy files
unlink("model_files", recursive = TRUE)

## End(Not run)
```

replace_abbreviations *Apply multiple string replacements to a data frame column.*

Description

This function iteratively applies a set of pattern-replacement pairs to the 'expression' column of a data frame. It performs multiple passes to ensure all replacements are applied, even if they depend on previous replacements. Patterns are treated as full words (or preceded by non-word characters and followed by non-word characters or an opening parenthesis) to prevent partial word matches.

Usage

```
replace_abbreviations(file, replacements, npass = 10)
```

Arguments

file	A data frame that must contain a column named expression.
replacements	A data frame with at least two columns: pattern (strings to be found) and replacement (strings to replace them with).
npass	The number of passes to apply the replacements. Multiple passes are useful if replacements can affect subsequent patterns.

Value

The input data frame file with the expression column modified by the replacements.

replace_values	<i>Replace values in a named vector while preserving order and other elements.</i>
----------------	--

Description

This function takes a source named vector (value) and a named vector of replacements. It replaces the values in value where the names match those in replacement, while keeping all other values and their original order intact.

Usage

```
replace_values(value, replacement)
```

Arguments

value	A named atomic vector (e.g., numeric, character, logical). This is the vector whose values will be modified. Must have names.
replacement	A named atomic vector of the same type as value (or coercible). The names in this vector specify which elements in value to replacement, and its values are the new values to be assigned. Must have names.

Value

A named vector with the updated values. The order and names of elements not specified in replacement remain unchanged from value.

Examples

```
# Example usage:
value <- c(a=2, b=3, d=5, foo=66)
replace <- c(foo=0, b=77)

new_value <- replace_values(value, replace)
print(new_value)
# Expected output: a=2, b=77, d=5, foo=0

# Example with some names in 'replacement' not present in 'value'
value2 <- c(x=10, y=20)
replacement2 <- c(y=50, z=100) # 'z' is not in src2
new_value2 <- replace_values(value2, replacement2)
print(new_value2)
# Expected output: x=10, y=50, with a warning about 'z'

# Example with missing names in value or replacement (will throw an error)
# tryCatch(replace_values(c(1,2), c(a=3)), error = function(e) message(e$message))
# tryCatch(replace_values(c(a=1, b=2), c(3,4)), error = function(e) message(e$message))
```

set_initval

Set or change initial values of the model states.

Description

This function updates the initial state variables of a rodeo model object. It performs a check to ensure that the provided initial values vector matches the defined state variables of the model in terms of names.

Usage

```
set_initval(model, init)
```

Arguments

model	A rodeo model object.
init	A named numeric vector representing the initial values for the model's state variables. The names of this vector must match the names of the model's state variables exactly.

Value

The modified rodeo model object with updated initial values.

set_pars	<i>Set or change parameter values in a model object.</i>
----------	--

Description

This function updates the parameters of a model object. It accepts a named vector of new parameters and uses the `replace_values` helper function to merge these into the existing or default parameters, preserving other values and their order.

Usage

```
set_pars(model, new_pars = NULL, default_pars = model$getPars())
```

Arguments

<code>model</code>	A model object that has <code>getPars()</code> and <code>setPars()</code> methods (e.g., a <code>rodeo</code> model object).
<code>new_pars</code>	A named vector of new parameter values to set. Its names should correspond to parameters in the model. If <code>NULL</code> , no parameters are changed beyond <code>default_pars</code> .
<code>default_pars</code>	An optional named vector representing the default parameters. If not provided, <code>model\$getPars()</code> will be used as the base onto which <code>new_pars</code> are applied.

Value

The modified model object with updated parameters.

See Also

[replace_values\(\)](#) [set_pars2\(\)](#)

set_pars2	<i>Set or change parameter values in a model object.</i>
-----------	--

Description

This function updates the parameters of a model object. You can provide new parameter values directly as named arguments via the ellipsis (`...`).

Usage

```
set_pars2(model, default_pars = model$getPars(), ...)
```

Arguments

<code>model</code>	A model object that has <code>getPars()</code> and <code>setPars()</code> methods.
<code>default_pars</code>	An optional named vector representing the default parameters. If not provided, <code>model\$getPars()</code> will be used.
<code>...</code>	Named arguments representing the new parameter values to set. These will override existing values in <code>default_pars</code> .

Details

This function works similar to `set_pars`, but `name = value`-pairs can be given directly, without an enclosing `c()`.

Value

The modified model object with updated parameters.

See Also

[replace_values\(\)](#) [set_pars\(\)](#)

Index

`generate_forcings`, [2](#)

`model_setup`, [3](#)

`replace_abbreviations`, [5](#)

`replace_values`, [5](#)

`replace_values()`, [7](#), [8](#)

`set_initval`, [6](#)

`set_pars`, [7](#)

`set_pars()`, [8](#)

`set_pars2`, [7](#)

`set_pars2()`, [7](#)