Package: rescomp (via r-universe)

June 2, 2026

Title Efficient Modelling of Resource Competition

Version 1.0.0

URL https://github.com/pyrrhicpachyderm/rescomp

Description Generate, simulate and visualise ODE models of consumer-resource interactions. At its core, 'rescomp' provides a resource competition modelling focused interface to 'deSolve', alongside flexible functions for visualising model properties and dynamics. More information, documentation and examples can be found on the package website.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports ggplot2, tidyr, deSolve, rlang, stats, cli, glue

Suggests rmarkdown, knitr, testthat (>= 3.0.0), vdiffr

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

Repository https://pyrrhicpachyderm.r-universe.dev

Date/Publication 2026-03-24 05:41:55 UTC

RemoteUrl https://github.com/pyrrhicpachyderm/rescomp

RemoteRef HEAD

RemoteSha 988c50e27872bff4b6e24b67ab83ea97de8a2e0c

Contents

`apply_event` *Applies a* `rescomp_event` *object to modify state variables*

Description

Applies the instantaneous changes specified by a `rescomp_event` object to the full set of state variables (species and resources).

Usage

```
apply_event(event_obj, species, resources, params, time)
```

Arguments

event_obj	An object of class rescomp_event.
species	A vector of species concentrations.
resources	A vector of resource concentrations.
params	A list of time-dependent parameters.
time	The current simulation time.

Details

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_event objects.

Value

A vector of state variables, species concentrations followed by resource concentrations.

Examples

```
batch_transfer <- event_batch_transfer(dilution = 0.1, resources = c(1, 1))
apply_event(
  batch_transfer,
  species = c(100, 200),
  resources = c(0.5, 2),
  params = list(),
  time = 0 # Not used by batch transfer events.
)
```

---

crmatrix *Shorthand to create a matrix with* spnum *rows and* resnum *columns*

Description

This produces an object that, when passed through rescomp functions such as funcresp_type1(), and ultimately provided to spec_rescomp(), inherits spnum and resnum from the arguments to spec_rescomp(). Thus, it is shorthand for matrix(c(...), nrow = spnum, ncol = resnum, byrow = TRUE). Inheritance only works when passed directly through rescomp functions; it will fail if used in the func provided to a function such as funcresp_custom().

Usage

```
crmatrix(..., byrow = TRUE)
```

Arguments

...	One or more numeric vectors that, concatenated by c(), produce a vector of length spnum * resnum.
byrow	A logical vector of length 1, equivalent to byrow of matrix(). Defaults to TRUE, unlike the byrow argument of matrix().

Value

S3 object of class rescomp_crmatrix.

Examples

```
pars <- spec_rescomp(
  spnum = 3, resnum = 2,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.10,
    0.08, 0.12,
    0.15, 0.05
  ))
)
plot_rescomp(sim_rescomp(pars))
```

---

def_cr_ode *Define consumer resource ODE function*

Description

Define consumer resource ODE function

Usage

```
def_cr_ode(t, y, pars)
```

Arguments

t	The current time of the simulation.
y	The vector of current estimates of consumers and resources in the simulation.
pars	S3 object of class rescomp containing model specification.

Value

List of length one, containing a vector of derivatives of y with respect to time.

event_batch_transfer *Define an event in which the species and resources are diluted by addition of new medium*

Description

Produces an event object representing a batch transfer by dilution with new medium. First, all existing species and resource concentrations are multiplied by `dilution`. Then, the resource concentrations listed in `resources`, multiplied by 1 - `dilution`, are added.

Usage

```
event_batch_transfer(dilution, resources)
```

Arguments

`dilution`	A numeric vector or `rescomp_coefs_vector` of length 1 (whose value should be between 0 and 1) representing the proportion of the original medium retained.
`resources`	A numeric vector or `rescomp_coefs_vector` of resource concentrations.

Value

S3 object of class `rescomp_event`.

Examples

```
m1 <- spec_rescomp(
  events = list(
    event_schedule_periodic(
      event_batch_transfer(dilution = 0.1, resources = 1),
      period = 250
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

event_res_add *Define an event in which a constant amount of some resources is added to (or removed from) the system*

Description

Produces an event object representing a pulse of added/removed resources.

Usage

```
event_res_add(resources, min_zero = TRUE)
```

Arguments

resources	A numeric vector or rescomp_coefs_vector of resource concentrations, by which the current resource concentrations are increased. Can be negative, to decrease resource concentrations.
min_zero	If this is TRUE, resulting resource concentrations are clamped to a minimum of zero. If this is FALSE, negative values of resources may reduce resource concentrations below zero.

Value

S3 object of class rescomp_event.

Examples

```
m1 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.08,
    0.06, 0.12
  )),
  events = list(
    event_schedule_periodic(
      event_res_add(c(-0.1, 0.1)),
      period = 250
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

---

event_res_custom *Define an event which instantaneously changes resource concentrations using an arbitrary function*

Description

Produces an event object suitable for building event schedules to pass spec_rescomp().

Usage

```
event_res_custom(func, resnum = NULL)
```

Arguments

func	A function that takes resources (a numeric vector of resource concentrations) and params (a list of parameters) and returns a numeric vector of resource concentrations.
resnum	The number of resources.

Details

If resnum is NULL, spec_rescomp() will attempt to infer it. This is fine if the result is used directly in an event schedule, but may fail if the result must be combined with other rescomp_events first.

Value

S3 object of class rescomp_event.

Examples

```
pars <- spec_rescomp(
  events = list(
    event_schedule_fixed(
      event = event_res_custom(function(res, params) {
        res * 2
      }),
      times = 500
    )
  )
)
plot_rescomp(sim_rescomp(pars))
```

---

event_res_mult *Define an event in which the resource populations are multiplied by some factor*

Description

Produces an event object a pulse of multiplicatively increased/decreased resource populations, e.g. a density-independent disturbance which kills prey species.

Usage

```
event_res_mult(resources_mult)
```

Arguments

resources_mult	A numeric vector or rescomp_coefs_vector by which to multiply the current resource concentrations. Should be greater than 1 for an increase, or less than 1 for a decrease. Should not be negative.

Value

S3 object of class rescomp_event.

Examples

```
m1 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.08,
    0.06, 0.12
  )),
  events = list(
    event_schedule_periodic(
      event_res_mult(c(0.5, 1.5)),
      period = 250
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

`event_schedule_fixed` *Define a schedule of an event occurring at a fixed set of times*

Description

Produces an event schedule object suitable for passing in the `events` argument to `spec_rescomp()`.

Usage

```
event_schedule_fixed(event_obj, times, priority = 0)
```

Arguments

`event_obj`	An object of class `rescomp_event`.
`times`	A numeric vector of times at which the event should occur.
`priority`	A number. If two events occur at the same time, the event with the lower priority number is processed first.

Value

S3 object of class `rescomp_event_schedule`.

Examples

```
m1 <- spec_rescomp(
  events = list(
    event_schedule_fixed(
      event_res_add(0.5),
      times = c(200, 400)
    ),
    event_schedule_fixed(
      event_sp_add(-500),
```

```
      times = c(600, 800)
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

`event_schedule_periodic` *Define a schedule of an event occurring repeatedly at a fixed period*

Description

Produces an event schedule object suitable for passing in the `events` argument to `spec_rescomp()`.

Usage

```
event_schedule_periodic(event_obj, period, start_time = period, priority = 0)
```

Arguments

`event_obj`	An object of class `rescomp_event`.
`period`	A number: the period at which the event occurs.
`start_time`	A number: the time at which the event first occurs.
`priority`	A number. If two events occur at the same time, the event with the lower priority number is processed first.

Value

S3 object of class `rescomp_event_schedule`.

Examples

```
m1 <- spec_rescomp(
  events = list(
    event_schedule_periodic(
      event_batch_transfer(dilution = 0.1, resources = 1),
      period = 250
    ),
    event_schedule_periodic(
      event_batch_transfer(dilution = 0.5, resources = 1),
      period = 250,
      start_time = 125
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

`event_set_introseq` *Define a schedule of one-off species introductions: one for each species.*

Description

Produces an event schedule object suitable for passing in the `events` argument to `spec_rescomp()`.

Usage

```
event_set_introseq(times, concentrations, priority = 0)
```

Arguments

`times`	A numeric vector, of length `spnum`, of times at which each species should be introduced.
`concentrations`	A numeric vector or `rescomp_coefs_vector`, of length `spnum`, of species concentrations; the concentration of each species that should be added at the corresponding time.
`priority`	A number. If two events occur at the same time, the event with the lower priority number is processed first.

Value

S3 object of class `rescomp_event_schedule`.

Examples

```
m1 <- spec_rescomp(
  spnum = 4, resnum = 4,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.05, 0.05, 0.05,
    0.05, 0.10, 0.05, 0.05,
    0.05, 0.05, 0.10, 0.05,
    0.05, 0.05, 0.05, 0.10
  )),
  events = list(
    event_schedule_periodic(
      event_batch_transfer(dilution = 0.5, resources = 1),
      period = 1000 / 16,
      priority = 0
    ),
    event_set_introseq(
      times = c(0, 250, 500, 750),
      concentrations = c(10, 100, 1000, 2000),
      priority = 1
    )
  ),
```

```
  cinit = 0
)
plot_rescomp(sim_rescomp(m1))
```

`event_sp_add` *Define an event in which a constant amount of some consumers is added to (or removed from) the system*

Description

Produces an event object representing a pulse of added/removed consumers, e.g. a species introduction.

Usage

```
event_sp_add(species, min_zero = TRUE)
```

Arguments

`species`	A numeric vector or `rescomp_coefs_vector` of species concentrations, by which the current species concentrations are increased. Can be negative, to decrease species concentrations.
`min_zero`	If this is TRUE, resulting species concentrations are clamped to a minimum of zero. If this is FALSE, negative values of `species` may reduce species concentrations below zero.

Value

S3 object of class `rescomp_event`.

Examples

```
m1 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.08,
    0.06, 0.12
  )),
  events = list(
    event_schedule_periodic(
      event_sp_add(c(-100, 100)),
      period = 250
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

`event_sp_custom` *Define an event which instantaneously changes species concentrations using an arbitrary function*

Description

Produces an event object suitable for building event schedules to pass `spec_rescomp()`.

Usage

```
event_sp_custom(func, spnum = NULL)
```

Arguments

`func`	A function that takes `species` (a numeric vector of species concentrations) and `params` (a list of parameters) and returns a numeric vector of species concentrations.
`spnum`	The number of species.

Details

If `spnum` is NULL, `spec_rescomp()` will attempt to infer it. This is fine if the result is used directly in an event schedule, but may fail if the result must be combined with other `rescomp_events` first.

Value

S3 object of class `rescomp_event`.

Examples

```
pars <- spec_rescomp(
  events = list(
    event_schedule_fixed(
      event = event_sp_custom(function(sp, params) {
        sp * 2
      }),
      times = 500
    )
  )
)
plot_rescomp(sim_rescomp(pars))
```

`event_sp_mult` *Define an event in which the consumer populations are multiplied by some factor*

Description

Produces an event object a pulse of multiplicatively increased/decreased consumer populations, e.g. a density-independent disturbance which kills consumer species.

Usage

```
event_sp_mult(species_mult)
```

Arguments

`species_mult`	A numeric vector or `rescomp_coefs_vector` by which to multiply the current species concentrations. Should be greater than 1 for an increase, or less than 1 for a decrease. Should not be negative.

Value

S3 object of class `rescomp_event`.

Examples

```
m1 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(crmatrix(
    0.10, 0.08,
    0.06, 0.12
  )),
  events = list(
    event_schedule_periodic(
      event_sp_mult(c(0.5, 1.5)),
      period = 250
    )
  )
)
plot_rescomp(sim_rescomp(m1))
```

frame_and_name *Convert object of class deSolve to a data frame and name columns*

Description

Convert object of class deSolve to a data frame and name columns

Usage

```
frame_and_name(model)
```

Arguments

model	List output from sim_rescomp(). First element is an object of class deSolve. Second element is an S3 object of class rescomp.

Value

data frame

Examples

```
pars <- spec_rescomp()
m1 <- sim_rescomp(pars)
frame_and_name(m1)
```

funcresp_custom *Define a functional response using an arbitrary function*

Description

Produces an object suitable to pass as the funcresp to spec_rescomp().

Usage

```
funcresp_custom(func, spnum = NULL, resnum = NULL)
```

Arguments

func	A function that takes resources (a numeric vector of resource concentrations) and params (a list of parameters) and returns a matrix of growth rates of each species on each resource.
spnum	The number of species; the number of rows in the matrix returned by func.
resnum	The number of resources; the expected length of the resources argument to func and the number of columns in the matrix returned by func.

Details

If spnum or resnum are NULL, spec_rescomp() will attempt to infer them. This is fine if the result is passed directly to spec_rescomp(), but may fail if the result must be combined with other rescomp_funcresps first.

Value

S3 object of class rescomp_funcresp.

Examples

```
# Type 1 functional response with fixed growth rates
m1 <- spec_rescomp(
  spnum = 2, resnum = 3,
  funcresp_custom(
    function(resources, params) {
      growth_rates <- c(0.2, 0.3)
      outer(growth_rates, resources)
    },
    spnum = 2
  )
)
plot_funcresp(m1)
```

---

funcresp_hill *Define a Hill functional response*

Description

Produces an object suitable to pass as the funcresp to spec_rescomp(). Creates a Hill functional response with maximum growth rate mumax, half saturation constant ks, and Hill coefficient n. mu_ij(R_j) = mumax_ij * (R_j)^n_ij / (ks_ij + (R_j)^n_ij) This an alternative parameterisation of a type 3 functional response, typically used with quota rather than efficiency.

Usage

```
funcresp_hill(mumax, ks, n)
```

Arguments

mumax	A matrix or rescomp_coefs_matrix, with one row per species and one column per resource. The maximum growth rate (if using quota) or maximum consumption rate (if using efficiency) of each species on each resource.
ks	A matrix or rescomp_coefs_matrix, with one row per species and one column per resource. The half saturation constant of each species for each resource; the resource concentration at which growth/consumption rate is half of mumax.
n	A matrix or rescomp_coefs_matrix, with one row per species and one column per resource. The Hill coefficient for each species on each resource. With n = 1, this is equivalent to a Monod functional response.

Value

S3 object of class `rescomp_funcresp`.

Examples

```
m1 <- spec_rescomp(
  spnum = 3,
  funcresp = funcresp_hill(
    mumax = crmatrix(0.2, 0.4, 0.4),
    ks = crmatrix(0.2, 1, 1),
    n = crmatrix(1, 1, 2)
  )
)
plot_funcresp(m1, maxx = 3)
```

---

funcresp_monod *Define a Monod functional response*

Description

Produces an object suitable to pass as the `funcresp` to `spec_rescomp()`. Creates a Monod functional response with maximum growth rate `mumax` and half saturation constant `ks`. mu_ij(R_j) = mumax_ij * R_j / (ks_ij + R_j) This an alternative parameterisation of a type 2 functional response, typically used with quota rather than efficiency.

Usage

```
funcresp_monod(mumax, ks)
```

Arguments

`mumax`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The maximum growth rate (if using quota) or maximum consumption rate (if using efficiency) of each species on each resource.
`ks`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The half saturation constant of each species for each resource; the resource concentration at which growth/consumption rate is half of mumax.

Value

S3 object of class `rescomp_funcresp`.

Examples

```
m1 <- spec_rescomp(
  spnum = 2,
  funcresp = funcresp_monod(
    mumax = crmatrix(0.2, 0.4),
    ks = crmatrix(0.2, 1)
  )
)
plot_funcresp(m1, maxx = 3)
```

`funcresp_type1` *Define a linear functional response*

Description

Produces an object suitable to pass as the `funcresp` to `spec_rescomp()`. Creates a linear or type 1 functional response with attack rate `a`. mu_ij(R_j) = a_ij * R_j

Usage

```
funcresp_type1(a)
```

Arguments

`a`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The attack rate (if using efficiency) or growth rate (if using quota) of each species on each resource.

Value

S3 object of class `rescomp_funcresp`.

Examples

```
m1 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(
    crmatrix(
      0.2, 0.3,
      0.4, 0.2
    )
  )
)
plot_funcresp(m1)

m2 <- spec_rescomp(
  spnum = 2, resnum = 2,
  funcresp = funcresp_type1(
    rescomp_coefs_lerp(
```

```
      crmatrix(
        0.2, 0.3,
        0.4, 0.2
      ),
      crmatrix(
        0.2, 0.3,
        0.1, 0.1
      ),
      "antibiotic_concentration"
    )
  ),
  params = list(antibiotic_concentration = rescomp_param_square())
)
plot_funcresp(m2, display_values = list(antibiotic_concentration = c(0, 0.5, 1)))
```

`funcresp_type2` *Define a Holling type 2 functional response*

Description

Produces an object suitable to pass as the `funcresp` to `spec_rescomp()`. Creates a Holling type 2 functional response with attack rate `a` and handling time `h`. mu_ij(R_j) = (a_ij * R_j) / (1 + a_ij * h_ij * R_j)

Usage

```
funcresp_type2(a, h)
```

Arguments

`a`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The maximum attack rate (if using efficiency) or growth rate (if using quota) of each species on each resource, when the resource is rare.
`h`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The handling time of each species for each resource, which reduces attack/growth rate when the resource becomes abundant. With h = 0, this is equivalent to a type 1 function response.

Value

S3 object of class `rescomp_funcresp`.

Examples

```
m1 <- spec_rescomp(
  spnum = 2,
  funcresp = funcresp_type2(
    a = crmatrix(0.4, 0.2),
    h = crmatrix(3, 1)
```

```
  )
)
plot_funcresp(m1, maxx = 3)
```

`funcresp_type3` *Define a Holling type 3 functional response*

Description

Produces an object suitable to pass as the `funcresp` to `spec_rescomp()`. Creates a Holling type 3 functional response with attack rate `a`, handling time `h`, and exponent `k`. mu_ij(R_j) = (a_ij * (R_j)^k_ij) / (1 + a_ij * h_ij * (R_j)^k_ij)

Usage

```
funcresp_type3(a, h, k)
```

Arguments

`a`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The maximum attack rate (if using efficiency) or growth rate (if using quota) of each species on each resource, when the resource is rare.
`h`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The handling time of each species for each resource, which reduces attack/growth rate when the resource becomes abundant.
`k`	A matrix or `rescomp_coefs_matrix`, with one row per species and one column per resource. The exponent (a.k.a. Hill coefficient) for each species on each resource. With k = 1, this is equivalent to a type 2 functional response. Each k must be greater than 1 for a true type 3 functional response.

Value

S3 object of class `rescomp_funcresp`.

Examples

```
m1 <- spec_rescomp(
  spnum = 3,
  funcresp = funcresp_type3(
    a = crmatrix(0.5, 0.3, 0.2),
    h = crmatrix(4, 2, 1),
    k = crmatrix(2)
  )
)
plot_funcresp(m1, maxx = 3)
```

get_coefs *Get coefficients from a vector/matrix or* rescomp_coefs_vector/rescomp_coefs_matrix *object*

Description

Provides a generic interface to a vector/matrix or a rescomp_coefs_vector/rescomp_coefs_matrix object. Called on a numeric vector/matrix (as appropriate to the function used), this will just return the vector/matrix. Called on a rescomp_coefs_vector/rescomp_coefs_matrix, this allows time-dependence via parameters. get_coefs() works on vector- or matrix-type objects, while the others work only on their specific types.

Usage

```
get_coefs(coefs_obj, params)

get_coefs_vector(coefs_obj, params)

get_coefs_matrix(coefs_obj, params)
```

Arguments

coefs_obj	A numeric vector/matrix or an object of class rescomp_coefs_vector/rescomp_coefs_matrix.
params	A list of time-dependent parameters.

Details

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_coefs objects.

Value

A vector/matrix of coefficients.

Examples

```
get_coefs(c(0.2, 0.3, 0.4), list())
get_coefs(matrix(c(0.2, 0.3, 0.4, 0.2), nrow = 2), list())
get_coefs_vector(c(0.2, 0.3, 0.4), list())
try(get_coefs_matrix(c(0.2, 0.3, 0.4), list()))

coefs <- rescomp_coefs_matrix_custom(
  function(params) {
    matrix(c(0.2, 0.3, 0.4, params$fourth_coeff), nrow = 2, ncol = 2)
  },
  nrow = 2, ncol = 2
)
get_coefs(coefs, list(fourth_coeff = 0.5))
```

get_event_times *Gets the list of times from a* rescomp_event_schedule *object*

Description

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_event_schedule objects.

Usage

```
get_event_times(event_schedule_obj, totaltime)
```

Arguments

event_schedule_obj	An object of class rescomp_event_schedule.
totaltime	The total time which the simulation will run for.

Value

A vector of times.

Examples

```
schedule1 <- event_schedule_periodic(
  event_batch_transfer(dilution = 0.1, resources = 1),
  period = 250
)
get_event_times(schedule1, 1000)
get_event_times(schedule1, 999)

schedule2 <- event_schedule_periodic(
  event_batch_transfer(dilution = 0.1, resources = 1),
  period = 250,
  start_time = 125
)
get_event_times(schedule2, 1000)
get_event_times(schedule2, 999)
```

get_funcresp *Get growth rates from a* rescomp_funcresp *object*

Description

Gets the growth rates of each species on each resource, given resource concentrations. These must be combined, according to whether the resouces are essential or substitutable, to get the overall growth rate for each species.

Usage

```
get_funcresp(funcresp_obj, spnum, resources, params)
```

Arguments

funcresp_obj	An object of class rescomp_funcresp.
spnum	The number of species.
resources	A vector of resource concentrations.
params	A list of time-dependent parameters.

Details

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_funcresp objects.

Value

A matrix of species growth rates on each resource, with spnum rows and length(resources) columns.

Examples

```
funcresp <- funcresp_custom(
  function(resources, params) {
    growth_rates <- params$scale * c(0.2, 0.3)
    outer(growth_rates, resources)
  },
  spnum = 2
)
get_funcresp(funcresp, 2, c(1, 4, 5, 6), list(scale = 2))
get_funcresp(funcresp, 2, 0.7, list(scale = 0.5))
try(get_funcresp(funcresp, 3, 0.7, list(scale = 0.5)))
```

get_params *Get params at an instant in time from a* rescomp_param *object*

Description

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_param objects.

Usage

```
get_params(param_obj, t)
```

Arguments

param_obj	An object of class rescomp_param.
t	The time at which to get the parameters.

Value

A numeric vector of length 1; the parameter value.

Examples

```
# Repeated pulsing and exponential decay
antibiotic_conc <- rescomp_param_custom(function(t) {
  0.5^(10 * (t %% 1))
})
get_params(antibiotic_conc, 0)
get_params(antibiotic_conc, 0.5)
get_params(antibiotic_conc, 1)

params <- list(r = 0.2, antibiotic_conc = antibiotic_conc)
get_params(params, 0.5)
```

get_ressupply *Get resource supply rates from a* rescomp_ressupply *object*

Description

Gets the resource supply rates of each resource, given the current resource concentrations.

Usage

```
get_ressupply(ressupply_obj, resources, params)
```

Arguments

ressupply_obj	An object of class rescomp_funcresp.
resources	A vector of resource concentrations.
params	A list of time-dependent parameters.

Details

This function is normally only for internal use, but is exported to aid users in debugging their created rescomp_ressupply objects.

Value

A vector of rates of change of resource concentrations, of the same length as resources.

Examples

```
# Two resources, A and B, with constant supply of A, and A spontaneously converting to B
ressupply <- ressupply_custom(
  function(resources, params) {
    conversion <- params$conversion * resources[1]
    return(c(params$supply - conversion, conversion))
  },
  resnum = 2
)
get_ressupply(ressupply, c(10, 20), list(supply = 3, conversion = 0.2))
try(get_ressupply(ressupply, c(10, 20, 30), list(supply = 3, conversion = 0.2)))
```

---

plot_funcresp *Plot functional responses*

Description

Plot functional responses

Usage

```
plot_funcresp(pars, maxx = 1, display_values, madj = FALSE)
```

Arguments

pars	S3 object of class rescomp detailing model parameters and specifications.
maxx	Numeric vector of length 1. Resource value to calculate per-capita growth rates up to (xlim).
display_values	Named list of vectors, with names matching names of pars$params. Each vector gives the values of the respective model parameters at which to plot the functional responses. Defaults are automatically inferred for most rescomp_param objects, but may be overwritten. rescomp_param_custom parameters must have their display values specified here if they were not defined at definition.

madj — Logical vector of length 1. Whether to standardise per capita growth rates by mortality.

Details

It is assumed that the funcresp of pars is constructed such that the growth rate of a species on a given resource depends only on the concentration of that resource, and not on other resources. This is the case for all built-in functional responses, but is not necessary the case if using funcresp_custom(). Plots are likely to be nonsensical or incorrect if this assumption is violated.

Value

A ggplot object.

Examples

```
pars <- spec_rescomp()
plot_funcresp(pars)

pars <- spec_rescomp(
  spnum = 2,
  resnum = 2,
  funcresp = funcresp_monod(
    mumax = crmatrix(
      0.7, 0.3,
      0.4, 0.5
    ),
    ks = crmatrix(1)
  )
)
plot_funcresp(pars)
plot_funcresp(pars, madj = TRUE)
plot_funcresp(pars, maxx = 5)

pars <- spec_rescomp(
  spnum = 2,
  funcresp = funcresp_type1(
    a = rescomp_coefs_lerp(
      crmatrix(0.12, 0.08),
      crmatrix(0.08, 0.12),
      param_name = "temperature"
    )
  ),
  params = list(temperature = rescomp_param_sine(period = 250))
)
plot_funcresp(pars)
plot_funcresp(pars, display_values = list(temperature = c(0.0, 0.25, 0.5, 0.75, 1.0)))
```

`plot_rescomp` *Plot consumer and resource dynamics from deSolve output*

Description

Plot consumer and resource dynamics from deSolve output

Usage

```
plot_rescomp(
  model,
  consumers = TRUE,
  resources = TRUE,
  logy = FALSE,
  lwd = 1,
  xlims = NULL
)
```

Arguments

`model`	List output from `sim_rescomp()`. First element is an object of class deSolve. Second element is an object of class rescomp.
`consumers`	Plot consumer dynamics? Default = TRUE.
`resources`	Plot resource dynamics? Default = TRUE.
`logy`	Log transfrom y-axis (default = FALSE).
`lwd`	Line width (default = 1)
`xlims`	Vector of length giving the time frame to plot.

Value

ggplot object

Examples

```
pars <- spec_rescomp()
m1 <- sim_rescomp(pars)
plot_rescomp(m1)
```

`rescomp_coefs_lerp` *Create a set of coefficients by linear interpolation*

Description

Produces an object that may be used in place of a vector or matrix in the creation of arguments to `spec_rescomp()`. Linearly interpolates between two sets of coefficients according to the value of a time-dependent parameter.

Usage

```
rescomp_coefs_lerp(coefs0, coefs1, param_name, param0 = 0, param1 = 1)
```

Arguments

`coefs0`, `coefs1`	Vectors, matrices, or objects of class `rescomp_coefs_vector` or `rescomp_coefs_matrix` to be interpolated between. Either both must be vectors, or both must be matrices.
`param_name`	A character vector of length 1; the name of the parameter to be interpolated according to.
`param0`, `param1`	The values of the parameter to take as the fixed points for interpolation.

Details

Coefficients are taken to have the values in `coef0` at `param = param0`, and the values in `coef1` at `param = param1`, and are linearly interpolated between. Each parameter in the vector/matrix is interpolated independently.

Value

S3 object of class `rescomp_coefs_vector` or `rescomp_coefs_matrix`, according to the types of `coefs0` and `coefs1`.

Examples

```
coefs_vec <- rescomp_coefs_lerp(
  c(0, 0, 0),
  c(2, 3, 5),
  "scale"
)
get_coefs(coefs_vec, list(scale = 0))
get_coefs(coefs_vec, list(scale = 0.5))
get_coefs(coefs_vec, list(scale = 1))
get_coefs(coefs_vec, list(scale = 1.5))

coefs_mat1 <- rescomp_coefs_lerp(
  matrix(c(0.2, 0.4, 0.3, 0.2), nrow = 2),
  matrix(c(0.2, 0.1, 0.3, 0.1), nrow = 2),
```

```
  "antibiotic1_concentration",
  param0 = 0,
  param1 = 200
)
get_coefs(coefs_mat1, list(antibiotic1_concentration = 0))
get_coefs(coefs_mat1, list(antibiotic1_concentration = 100))
get_coefs(coefs_mat1, list(antibiotic1_concentration = 200))

coefs_mat2 <- rescomp_coefs_lerp(
  coefs_mat1,
  matrix(c(0.0, 0.0, 0.0, 0.0), nrow = 2),
  "antibiotic2_concentration",
  param0 = 0,
  param1 = 100
)
get_coefs(coefs_mat2, list(antibiotic1_concentration = 100, antibiotic2_concentration = 50))
```

`rescomp_coefs_matrix_custom` *Create a matrix of coefficients using an arbitrary function*

Description

Produces an object that may be used in place of a matrix in the creation of arguments to `spec_rescomp()`.

Usage

```
rescomp_coefs_matrix_custom(func, nrow, ncol)
```

Arguments

`func`	A function that takes `params` (a list of parameters) and returns a matrix.
`nrow`	The number of rows in the matrix returned by `func`.
`ncol`	The number of columns in the matrix returned by `func`.

Value

S3 object of class `rescomp_coefs_matrix`.

Examples

```
# Matrix with one time-varying coefficient
rescomp_coefs_matrix_custom(
  function(params) {
    matrix(c(0.2, 0.3, 0.4, params$fourth_coeff), nrow = 2, ncol = 2)
  },
  nrow = 2, ncol = 2
)
```

rescomp_coefs_vector_custom

Create a vector of coefficients using an arbitrary function

Description

Produces an object that may be used in place of a vector in the creation of arguments to `spec_rescomp()`.

Usage

```
rescomp_coefs_vector_custom(func, length)
```

Arguments

`func`	A function that takes `params` (a list of parameters) and returns a vector.
`length`	The length of the vector returned by `func`.

Value

S3 object of class `rescomp_coefs_vector`.

Examples

```
# Vector with one time-varying coefficient
rescomp_coefs_vector_custom(
  function(params) {
    c(0.2, 0.3, 0.4, params$fourth_coeff)
  },
  length = 4
)
```

rescomp_param_custom

Create a rescomp parameter using an arbitrary function

Description

Produces an object suitable to include in a list of rescomp parameters, providing time-dependence.

Usage

```
rescomp_param_custom(func, display_values = NULL)
```

Arguments

`func`	A function that takes `t` (time) and returns a number to use as a parameter.
`display_values`	A numeric vector of values of the parameter to use in `plot_funcresp()`. If not specified here, must be specified in `plot_funcresp()`.

Value

S3 object of class `rescomp_param`.

Examples

```
# Repeated pulsing and exponential decay
antibiotic_conc <- rescomp_param_custom(function(t) {
  0.5^(10 * (t %% 1))
})
get_params(antibiotic_conc, 0)
get_params(antibiotic_conc, 0.5)
get_params(antibiotic_conc, 1)
```

`rescomp_param_sine` *Create a rescomp parameter using a sine/square/triangle wave*

Description

Produces an object suitable to include in a list of rescomp parameters, providing time-dependence. Triangle and square waves are phase-shifted to be similiar in shape to a sine wave with the same period and offset, such that the peaks and troughs occur in the same places.

Usage

```
rescomp_param_sine(
  period = 1,
  min = 0,
  max = 1,
  offset = 0,
  display_values = c(min, max)
)

rescomp_param_triangle(
  period = 1,
  min = 0,
  max = 1,
  offset = 0,
  display_values = c(min, max)
)

rescomp_param_square(
  period = 1,
  min = 0,
  max = 1,
  offset = 0,
  display_values = c(min, max)
)
```

Arguments

period	The period of the wave.
min	The minimum value of the parameter; the mean minus the amplitude.
max	The maximum value of the parameter; the mean plus the amplitude.
offset	The phase shift of the wave. For a sine or triangle wave the time at which its value is equal to the mean and increasing. For a square wave, the time at which it increases to the maximum value.
display_values	A numeric vector of values of the parameter to use in plot_funcresp().

Value

S3 object of class rescomp_param.

Examples

```
sine <- rescomp_param_sine(period = 1)
cosine <- rescomp_param_sine(period = 1, offset = -0.25)
triangle <- rescomp_param_triangle(period = 1)
square <- rescomp_param_square(period = 1)

times <- seq(from = 0, to = 2, by = 0.01)
plot(times, get_params(sine, times), type = "l", col = "black")
lines(times, get_params(cosine, times), col = "blue")
lines(times, get_params(triangle, times), col = "maroon3")
lines(times, get_params(square, times), col = "orange")
```

---

ressupply_chemostat *Create a resource supply rate using chemostat dynamics*

Description

Produces an object suitable to pass as the ressupply to spec_rescomp().

Usage

```
ressupply_chemostat(dilution, concentration)
```

Arguments

dilution	A numeric vector or rescomp_coefs_vector, of length one.
concentration	A vector or rescomp_coefs_vector, with one number per resource. The concentration of each resource in the incoming medium.

Value

S3 object of class rescomp_ressupply.

Examples

```
ressupply <- ressupply_chemostat(
  dilution = 0.01,
  concentration = rescomp_coefs_lerp(c(0, 0, 0), c(2, 3, 4), "ressupply_scaling")
)
get_ressupply(ressupply, c(2, 4, 10), list(ressupply_scaling = 0))
get_ressupply(ressupply, c(3, 3, 3), list(ressupply_scaling = 1))
```

---

ressupply_constant *Create a resource supply rate using a constant rate resource supply*

Description

Produces an object suitable to pass as the `ressupply` to `spec_rescomp()`.

Usage

```
ressupply_constant(rate)
```

Arguments

`rate`	A vector or `rescomp_coefs_vector`, with one number per resource. The supply rate of each resource.

Value

S3 object of class `rescomp_ressupply`.

Examples

```
ressupply <- ressupply_constant(c(0.2, 0.3))
get_ressupply(ressupply, c(2, 10), list())
get_ressupply(ressupply, c(5, 20), list())
# The above two give the same result; constant supply doesn't depend on existing concentration.

ressupply <- ressupply_constant(rescomp_coefs_lerp(c(0.2, 0.3), c(0.4, 0.6), "extra_supply"))
get_ressupply(ressupply, c(2, 10), list(extra_supply = 0.2))
get_ressupply(ressupply, c(2, 10), list(extra_supply = 0.8))
```

ressupply_custom *Create a resource supply rate using an arbitrary function*

Description

Produces an object suitable to pass as the ressupply to spec_rescomp().

Usage

```
ressupply_custom(func, resnum = NULL)
```

Arguments

func	A function that takes resources (a numeric vector of resource concentrations) and params (a list of parameters) and returns a vector of rates of change of each resource.
resnum	The number of resources; the expected length of the resources argument to func and the length of the vector returned by func.

Details

If resnum is NULL, spec_rescomp() will attempt to infer it. This is fine if the result is passed directly to spec_rescomp(), but may fail if the result must be combined with other rescomp_ressupply first.

Value

S3 object of class rescomp_ressupply.

Examples

```
# Two resources, A and B, with constant supply of A, and A spontaneously converting to B
ressupply <- ressupply_custom(
  function(resources, params) {
    conversion <- params$conversion * resources[1]
    return(c(params$supply - conversion, conversion))
  },
  resnum = 2
)
get_ressupply(ressupply, c(10, 20), list(supply = 3, conversion = 0.2))
```

ressupply_logistic *Create a resource supply rate using logistic resource growth*

Description

Produces an object suitable to pass as the `ressupply` to `spec_rescomp()`.

Usage

```
ressupply_logistic(r, k)
```

Arguments

r	A vector or `rescomp_coefs_vector`, with one number per resource. The intrinsic growth rate of each resource.
k	A vector or `rescomp_coefs_vector`, with one number per resource. The carrying capacity of each resource.

Value

S3 object of class `rescomp_ressupply`.

Examples

```
ressupply <- ressupply_logistic(
  r = rescomp_coefs_lerp(c(0.2, 0.3), c(0, 0), "growth_inhibition"),
  k = c(10, 20)
)
get_ressupply(ressupply, c(2, 10), list(growth_inhibition = 0))
get_ressupply(ressupply, c(20, 0), list(growth_inhibition = 0))
get_ressupply(ressupply, c(2, 10), list(growth_inhibition = 0.8))
```

sim_rescomp *Simulate resource competition (a convenience wrapper for `deSolve::ode()`)*

Description

Simulate resource competition (a convenience wrapper for `deSolve::ode()`)

Usage

```
sim_rescomp(pars, totaltime, cinit, rinit, ...)
```

Arguments

pars	S3 object of class rescomp returned by rescomp::spec_rescomp().
totaltime	Numeric vector of length 1: the total simulation time. If provided, overrides the value in pars.
cinit	Numeric vector of length 1 or length spnum specifying initial consumer state values (densities). If provided, overrides the value in pars.
rinit	Numeric vector of length 1 or length resnum specifying initial resource state values (concentrations). If provided, overrides the value in pars.
...	Other arguments passed to deSolve::ode()

Value

A list of two comprising i) the model dynamics and ii) model specifications.

Examples

```
pars <- spec_rescomp()
results1 <- sim_rescomp(pars = pars)
plot_rescomp(results1)

results2 <- sim_rescomp(pars = pars, totaltime = 100, cinit = 1000)
plot_rescomp(results2)
```

---

spec_rescomp *Generate list of parameters for a consumer-resource model to be passed to* sim_rescomp()

Description

Generate list of parameters for a consumer-resource model to be passed to sim_rescomp()

Usage

```
spec_rescomp(
  spnum = 1,
  resnum = 1,
  funcresp = funcresp_type1(crmatrix(0.1)),
  quota = crmatrix(0.001),
  efficiency,
  essential = FALSE,
  mort = 0.03,
  ressupply = ressupply_chemostat(0.03, 1),
  params = list(),
  events = list(),
  totaltime = 1000,
  cinit = 10,
```

```
  rinit = 1,
  verbose = FALSE
)
```

Arguments

spnum	Integer vector of length 1: the number of consumers.
resnum	Integer vector of length 1: the number of resources.
funcresp	An object of class rescomp_funcresp specifying the functional response.
quota	Numeric matrix or rescomp_coefs_matrix, the elements of which give the resource quotas. The number of rows and columns should be equal to spnum and resnum respectively. Mutually exclusive with efficiency.
efficiency	Numeric matrix or rescomp_coefs_matrix, the elements of which give the efficiency of each consumer on each resource. The number of rows and columns should be equal to spnum and resnum respectively. Mutually exclusive with quota.
essential	Logical vector of length 1. If FALSE resources are substitutable.
mort	Numeric vector or rescomp_coefs_vector of length spnum, specifying density independent mortality rates.
ressupply	An object of class rescomp_ressupply specifying the resource supply.
params	A list specifying a set of parameters which may vary with time, on which other parameters of the model (e.g. funcresp, ressupply) may depend. rescomp_param objects in the list are parsed to allow time dependence, while other objects are passed directly.
events	A list of objects of class rescomp_event_schedule, specifying events that instantaneously change consumer or resource densities.
totaltime	Numeric vector of length 1: the total simulation time.
cinit	Numeric vector of length 1 or length spnum specifying initial consumer state values (densities).
rinit	Numeric vector of length 1 or length resnum specifying initial resource state values (concentrations).
verbose	If TRUE (default) prints model and simulation summary specs.

Details

Only one of efficiency and quota should be specified. Specifying both is an error. The default, if neither is specified, is to use quota. If using quota, the functional responses are taken to give per capita growth rates. If using efficiency, the functional responses are taken to give attack rates.

Value

S3 object of class rescomp.

Examples

```
# Using default parameters.
m1 <- spec_rescomp()
plot_rescomp(sim_rescomp(m1))

# With two species, two resources, and type 2 functional responses.
m2 <- spec_rescomp(
  spnum = 2,
  resnum = 2,
  funcresp = funcresp_type2(
    a = crmatrix(
      0.1, 0.2,
      0.15, 0.15
    ),
    h = crmatrix(1)
  ),
  totaltime = 500
)
plot_rescomp(sim_rescomp(m2))

# With serial dilution (batch transfer) events.
m3 <- spec_rescomp(
  ressupply = ressupply_constant(0),
  mort = 0,
  events = list(
    event_schedule_periodic(
      event_batch_transfer(dilution = 0.1, resources = 1),
      period = 125
    )
  ),
  totaltime = 500
)
plot_rescomp(sim_rescomp(m3))

# Growth rates affected by seasonal temperature fluctuations.
m4 <- spec_rescomp(
  spnum = 2,
  funcresp = funcresp_type1(
    a = rescomp_coefs_lerp(
      crmatrix(0.12, 0.08),
      crmatrix(0.08, 0.12),
      param_name = "temperature"
    )
  ),
  params = list(temperature = rescomp_param_sine(period = 125)),
  totaltime = 500
)
plot_rescomp(sim_rescomp(m4))
```

Index