Package 'gadget3'

May 8, 2026

Type Package

Title Globally-Applicable Area Disaggregated General Ecosystem Toolbox V3

Version 0.15-1

Date 2026-01-12

Maintainer Jamie Lentin <lentinj@shuttlethread.com>

Description A framework to assist creation of marine ecosystem models, generating either 'R' or 'C++' code which can then be optimised using the 'TMB' package and standard 'R' tools. Principally designed to reproduce gadget2 models in 'TMB', but can be extended beyond gadget2's capabilities.
Kasper Kristensen, Anders Nielsen, Casper W. Berg, Hans Skaug, Bradley M. Bell (2016) <doi:10.18637/jss.v070.i05> ``TMB: Automatic Differentiation and Laplace Approximation.''.
Begley, J., & Howell, D. (2004) <https://files01.core.ac.uk/download/pdf/225936648.pdf> ``An overview of Gadget, the globally applicable area-disaggregated general ecosystem toolbox. ICES.''.

URL https://gadget-framework.github.io/gadget3/, https://github.com/gadget-framework/gadget3/

Encoding UTF-8

Depends R (>= 4.2.0)

Imports digest, rlang (>= 0.4.5), stats, TMB (>= 1.7.0), utils,

Suggests dplyr, knitr, magrittr (>= 1.5), rmarkdown, unittest (>= 1.4)

VignetteBuilder knitr

License GPL-2

RoxygenNote 7.0.2

NeedsCompilation no

Author Jamie Lentin [aut, cre] (ORCID: <https://orcid.org/0000-0001-5727-2996>), Bjarki Thor Elvarsson [aut] (ORCID: <https://orcid.org/0000-0001-5855-1188>),

William Butler [aut] (ORCID: <https://orcid.org/0000-0002-3286-0748>),
Marine and Freshwater Research Institute (Iceland) [cph]

Repository CRAN

Date/Publication 2026-01-12 14:00:02 UTC

Contents

`aaa_lang` *Gadget3 language utilities*

Description

Produce objects with special meaning to gadget3

Usage

```
g3_native(r, cpp, depends = c())
g3_global_formula(f = quote(noop), init_val = NULL)
```

Arguments

`r`	An R function to decorate with a 'C++' equivalent
`cpp`	Either: 1. A character string containing the 'C++' equivalent as a Lambda function 2. A character string containing 'C++' function template definition, calling the function `__fn__` 3. A list of type-casts to use when calling an equivalently named native function
`depends`	A list of string names of dependent functions. The content of this and the initial `[]` for any Lambda function should match.
`f`	An optional formula to modify the content of a globablly-defined variable
`init_val`	An optiona formula to set the initial value of a globally-defined variable

Details

These functions are generally for gadget3 development, but made available so actions can be produced outside the package.

Value

g3_native: Returns a function that can be used in formulas for both R and TMB-based models.

g3_global_formula: Returns a formula that will be defined globally, and this can preserve state across timesteps.

Examples

```
# The definition of g3_env$ratio_add_pop looks like:
eg_ratio_add_pop <- g3_native(r = function(orig_vec, orig_amount,
                                           new_vec, new_amount) {
    ((orig_vec * orig_amount + new_vec * new_amount)
      /
    avoid_zero_vec(orig_amount + new_amount))
}, cpp = '[](vector<Type> orig_vec, vector<Type> orig_amount,
                            vector<Type> new_vec, vector<Type> new_amount)
                            -> vector<Type> {
    return (orig_vec * orig_amount + new_vec * new_amount)
             /
           avoid_zero_vec(orig_amount + new_amount);
}', depends = c('avoid_zero_vec'))
# eg_ratio_add_pop() can then be used in formulas, both in R & TMB.

# Define a random walk action, using g3_global_formula to keep track of
# previous value. NB: my_randomwalk_prevrec must be unique in a model
random_walk_action <- g3_formula(quote({
    if (cur_time > 0) nll <- nll + dnorm(x, stock__prevrec, 1, 1)
    my_randomwalk_prevrec <- x
}), x = 'TODO', my_randomwalk_prevrec = g3_global_formula(init_val = 0.0))
```

---

aab_env *Gadget3 global environment*

Description

Functions available to any gadget3 model

Details

`g3_env` is the top-level environment that any gadget3 model uses, populated with utility functions.

NB: Several functions have `_vec` variants. Due to TMB limitations these should be used when you have a vector not scalar input.

ADREPORT

TMB's `ADREPORT` function. See sdreport documentation

as_integer

C++ compatible equivalent to `as.integer`

as.numeric

R `as.numeric` or TMB `asDouble`

assert_msg

C++/R function that ensures expression is true, or stops model.

```
assert_msg(x > 0, "x must be positive")
```

avoid_zero

Adds small value to input to ensure output is never zero `avoid_zero_vec` is identical to `avoid_zero`, and is only present for backward compatibility.

bounded / bounded_vec

Ensures *x* is within limits *b* & *a*.

If *x* positive, return *a*. If *x* negative, *b*. If *x* `-10..10`, smoothly transition from *b* to *a*

```
bounded_vec(x, 200, 100)
```

g3_matrix_vec

Apply matrix transformation *tf* to vector *vec*, return resultant vector.

```
g3_matrix_vec(tf, vec)
```

lgamma_vec

Vector equivalent of `lgamma`

logspace_add

TMB's `logspace_add`, essentially a differentiable version of `pmax`.

normalize_vec

Divide vector *a* by it's sum, i.e. so it now sums to 1. If vector is all-zero, return an all-zero vector instead.

nvl

Return first non-null argument. NB: No C++ implementation.

print_array

Utility to pretty-print array *ar*

ratio_add_pop

Sum *orig_vec* & *new_vec* according to ratio of *orig_amount* & *new_amount*

nonconform_add / nonconform_mult / nonconform_div / nonconform_div_avz

Scalar sum/multiply/divide 2 non-conforming arrays, by treating the latter as a vector and repeating as many times as required. Results will be structured identically to the first array.

`nonconform_div_avz(x, y)` is equivalent to `nonconform_div(x, avoid_zero_vec(y))`

REPORT

TMB's `REPORT` function.

REprintf

Equivalent of RCpp REprintf

Rprintf

Equivalent of RCpp Rprintf

Examples

```
## avoid_zero
g3_eval(quote( c( avoid_zero(0), avoid_zero(10) ) ))
g3_eval(quote( avoid_zero(0:5) ))

## bounded / bounded_vec
curve(g3_eval(quote( bounded(x, 200, 100) ), x = x), -100, 100)

## logspace_add
curve(g3_eval(quote( logspace_add(x, 10) ), x = x), 0, 40)

## normalize_vec
g3_eval(quote( normalize_vec(c( 4, 4, 8, 2 )) ))

## nonconform_mult
g3_eval(quote( nonconform_mult(
    array(seq(0, 4*5*6), dim = c(4,5,6)),
    c(1e1, 1e2, 1e3, 1e4)) ))

## nonconform_div_avz
g3_eval(quote( nonconform_div_avz(
    array(seq(0, 4*5*6), dim = c(4,5,6)),
    c(1e1, 1e2, 0, 1e4)) ))
g3_eval(quote( nonconform_div(
    array(seq(0, 4*5*6), dim = c(4,5,6)),
    avoid_zero(c(1e1, 1e2, 0, 1e4))) ))
```

action_age *Gadget3 age action*

Description

Add ageing actions to a g3 model

Usage

```
g3a_age(
        stock,
        output_stocks = list(),
        output_ratios = rep(1/length(output_stocks),
        times = length(output_stocks)),
        run_f = ~cur_step_final,
        run_at = g3_action_order$age,
        transition_at = g3_action_order$age)
```

Arguments

stock	g3_stock to age.
output_stocks	List of g3_stocks that oldest specimens in *stock* should move into.
output_ratios	Vector of proportions for how to distribute into *output_stocks*, default evenly spread.
run_f	formula specifying a condition for running this action, default is end of model year.
run_at	Integer order that actions will be run within model, see g3_action_order.
transition_at	Integer order that transition actions will be run within model, see g3_action_order.

Value

An action (i.e. list of formula objects) that will, for the given *stock*...

1. Move the final age group into temporary storage, stock__transitioning_num / stock__transitioning_wgt
2. Move the contents of all other age groups into the age group above
3. Move the contents of the temporary storage into *output_stocks*

If *stock* has only one age, and *output_stocks* has been specified, then the contentes will be moved, if *output_stocks* is empty, then the action will do nothing.

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockmature, g3_stock

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock('ling_mat', seq(20, 156, 4)) %>% g3s_age(5, 15)

# Ageing for immature ling
age_action <- g3a_age(ling_imm,
    output_stocks = list(ling_mat))
```

`action_grow` *Gadget3 growth action*

Description

Add growth/maturity actions to a g3 model

Usage

```
g3a_grow_lengthvbsimple(
        linf_f = g3_parameterized('Linf', by_stock = by_stock),
        kappa_f = g3_parameterized('K', by_stock = by_stock),
        by_stock = TRUE)

g3a_grow_weightsimple(
        alpha_f = g3_parameterized('walpha', by_stock = by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = by_stock),
        by_stock = TRUE)

g3a_grow_impl_bbinom(
        delta_len_f = g3a_grow_lengthvbsimple(by_stock = by_stock),
        delta_wgt_f = g3a_grow_weightsimple(by_stock = by_stock),
        beta_f = g3_parameterized('bbin', by_stock = by_stock),
        maxlengthgroupgrowth,
        by_stock = TRUE)

g3a_grow_length_multspec(
        p0 = g3_parameterized('multispec.p0', value = 1, by_stock = by_stock),
        p1 = g3_parameterized('multispec.p1', value = 1, by_stock = by_stock),
        p2 = g3_parameterized('multispec.p2', value = 1, by_stock = by_stock),
        p3 = g3_parameterized('multispec.p3', value = 0, by_stock = by_stock),
        temperature = 0,
        by_stock = TRUE)

g3a_grow_weight_multspec(
        p4 = g3_parameterized('multispec.p4', value = 1, by_stock = by_stock),
        p5 = g3_parameterized('multispec.p5', value = 1, by_stock = by_stock),
        p6 = g3_parameterized('multispec.p6', value = 0, by_stock = by_stock),
        p7 = g3_parameterized('multispec.p7', value = 1, by_stock = by_stock),
```

```
        p8 = g3_parameterized('multispec.p8', value = 0, by_stock = by_stock),
        temperature = 0,
        by_stock = TRUE)

g3a_grow_length_weightjones(
       p0 = g3_parameterized('weightjones.p0', value = 0, by_stock = by_stock),
       p1 = g3_parameterized('weightjones.p1', value = 0, by_stock = by_stock),
       p2 = g3_parameterized('weightjones.p2', value = 1, by_stock = by_stock),
       p3 = g3_parameterized('weightjones.p3', value = 0, by_stock = by_stock),
       p4 = g3_parameterized('weightjones.p4', value = 1, by_stock = by_stock),
      p5 = g3_parameterized('weightjones.p5', value = 100, by_stock = by_stock),
       p6 = g3_parameterized('weightjones.p6', value = 1, by_stock = by_stock),
       p7 = g3_parameterized('weightjones.p7', value = 1, by_stock = by_stock),
       reference_weight = 0,
       temperature = 0,
       by_stock = TRUE)

g3a_grow_weight_weightjones(
       q0 = g3_parameterized('weightjones.q0', value = 1, by_stock = by_stock),
       q1 = g3_parameterized('weightjones.q1', value = 1, by_stock = by_stock),
       q2 = g3_parameterized('weightjones.q2', value = 1, by_stock = by_stock),
       q3 = g3_parameterized('weightjones.q3', value = 1, by_stock = by_stock),
       q4 = g3_parameterized('weightjones.q4', value = 1, by_stock = by_stock),
       q5 = g3_parameterized('weightjones.q5', value = 0, by_stock = by_stock),
      max_consumption = g3a_predate_maxconsumption(temperature = temperature),
       temperature = 0,
       by_stock = TRUE)

g3a_growmature(stock, impl_f, maturity_f = ~0, output_stocks = list(),
    output_ratios = rep(1/length(output_stocks), times = length(output_stocks)),
    transition_f = ~cur_step_final, run_f = ~TRUE,
    run_at = g3_action_order$grow,
    transition_at = g3_action_order$mature)
```

Arguments

`linf_f`	A formula to substitute for L_∞.
`kappa_f`	A formula to substitute for κ.
`alpha_f`	A formula to substitute for α.
`beta_f`	A formula to substitute for β.
`p0`, `p1`, `p2`, `p3`, `p4`, `p5`, `p6`, `p7`, `p8`, `q0`, `q1`, `q2`, `q3`, `q4`, `q5`	A formula to substitute for the equivalent value.
`max_consumption`	Maximum predator consumption, see `g3a_predate_maxconsumption`.
`temperature`	A formula providing values for the current temperature, likely implemented with `g3_timeareadata`.

`maxlengthgroupgrowth`
: An integer with the maximum length groups an individual can jump in one step.

`reference_weight`
: Reference weight. see formula for `g3a_grow_length_weightjones`.

`stock`
: `g3_stock` to grow.

`delta_len_f`
: A formula defining a non-negative vector for mean increase in length for `stock` for each lengthgroup, as defined by `g3a_grow_lengthvbsimple`.

`delta_wgt_f`
: A formula defining the corresponding weight increase as a matrix of lengthgroup to lengthgroup delta for `stock`, as defined by `g3a_grow_weightsimple`.

`by_stock`
: Change the default parameterisation (e.g. to be by 'species'), see `g3_parameterized`.

`impl_f`
: A pair of formula objects, as defined by `g3a_grow_impl_bbinom`. Both define a matrix of length groups `i` to length group deltas `j` (0..*maxlengthgroupgrowth*), the values in the first indicate the proportion of individuals moving from `i` to `i + j`, the values in the second indicate the corresponding weight increase of individuals moving from `i` to `i + j`.

`maturity_f`
: A maturity formula, as defined by `g3a_mature_constant`.

`output_stocks`
: List of `g3_stock`s that maturing *stock* should move into.

`output_ratios`
: Vector of proportions for how to distribute into *output_stocks*, summing to 1, default evenly spread.

`transition_f`
: formula specifying a contition for running maturation steps as well as growth, default final step of year.

`run_f`
: formula specifying a condition for running this action, default always runs.

`run_at`
: Integer order that actions will be run within model, see `g3_action_order`.

`transition_at`
: Integer order that transition actions will be run within model, see `g3_action_order`.

Details

A model can have any number of `g3a_growmature` actions, so long as the calling arguments are different. For instance, `run_f = ~age == 5` and `run_f = ~age == 7`.

impl_f's dependent variables are analysed to see what will affect growth. If nothing but `cur_step_size` will affect growth, then growth will only be recalculated when the step size changes.

Value

g3a_grow_lengthvbsimple: Returns a formula for use as *delta_len_f*:

$$\Delta L_i = (L_\infty - L_i)(1 - e^{-\kappa \Delta t})$$

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

g3a_grow_weightsimple: Returns a formula for use as *delta_wgt_f*:

$$\Delta W_{i,j} = \alpha((L_i + \Delta L_j)^\beta - {L_i}^\beta)$$

ΔL Vector of all possible length group increases i.e `0..maxlengthgroupgrowth`

g3a_grow_length_multspec: Returns a formula for use as *delta_len_f*:

$$\Delta L_i = \Delta t p_0 L_i^{p_1} \psi_i (p_2 T + p_3)$$

p_x Supplied parameters
Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`
L_i Current length
ψ_i Feeding level of stock. See `g3a_predate_catchability_predator`
T Temperature of current region

g3a_grow_weight_multspec: Returns a formula for use as *delta_wgt_f*:

$$\Delta W_{i,j} = \Delta t p_4 {W_i}^{p_5} (\psi_i - p_6)(p_7 T + p_8)$$

p_x Supplied parameters
Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`
W_i Current mean weight
ψ_i Feeding level of stock. See `g3a_predate_catchability_predator`
T Temperature of current region

Note that the equation is not dependent on the change in length, the value will be the same for each j.

g3a_grow_length_weightjones: Returns a formula for use as *delta_len_f*:

$$r = \frac{W_i - (p_0 + \psi_i (p_1 + p_2 \psi_i)) W_{ref}}{W_i}$$

$$\Delta L_i = minmax(p_3 + p_4 r, 0, p_5) \frac{\Delta W_{i,j}}{p_6 p_7 {L_i}^{(p_7 - 1)}}$$

W_i Current mean weight
p_x Supplied parameters
ψ_i Feeding level of stock. See `g3a_predate_catchability_predator`
W_{ref} Reference weight, from the *reference_weight* parameter
$\Delta W_{i,j}$ Change in weight, i.e. the output from the *delta_wgt_f* formula, probably `g3a_grow_weight_weightjones`.

g3a_grow_weight_weightjones: Returns a formula for use as *delta_wgt_f*:

$$\Delta W_{i,j} = \Delta t (\frac{M \psi_i}{q_0 {W_i}^{q_1}} - q_2 {W_i}^{q_3} e^{(q_4 T + q_5)})$$

q_x Supplied parameters
Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`
M Maximum theoretical consumption, as defined by `g3a_predate_maxconsumption`
ψ_i Feeding level of stock. See `g3a_predate_catchability_predator`
W_i Current mean weight
T Temperature of current region

Note that the equation is not dependent on the change in length, the value will be the same for each j.

g3a_grow_impl_bbinom: formula object converting mean growths using beta-binomia distribution. See https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#beta-binomial

g3a_growmature: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Move any maturing individuals into temporary storage, `stock__transitioning_num`/`stock__transitioning_wgt`
2. Calculate increase in length/weight using *growth_f* and *impl_f*
3. Move the contents of the temporary storage into *output_stocks*

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockgrowth, g3_stock

Examples

```
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4))
ling_mat <- g3_stock(c(species = 'ling', 'mat'), seq(20, 156, 4))

# Growth / maturity for immature ling
growth_action <- g3a_growmature(ling_imm,
    impl_f = g3a_grow_impl_bbinom(
        # Parameters will be ling.Linf, ling.K
        g3a_grow_lengthvbsimple(by_stock = 'species'),
        # Parameters will be ling_imm.walpha, ling_imm.wbeta
        g3a_grow_weightsimple(),
        maxlengthgroupgrowth = 15),
    maturity_f = g3a_mature_constant(
        alpha = g3_parameterized('ling.mat1', scale = 0.001),
        l50 = g3_parameterized('ling.mat2')),
        output_stocks = list(ling_mat))

# Multspec growth - define a data frame with temperature
temperature <- g3_timeareadata(
    'temp',
    data.frame(year = 2000, step=c(1,2), temp=c(10, 14)),
    value_field = "temp" )

ms_growth_actions <- list(
  g3a_growmature(ling_imm, g3a_grow_impl_bbinom(
    g3a_grow_length_multspec(temperature = temperature),
    g3a_grow_weight_multspec(temperature = temperature),
    maxlengthgroupgrowth = 8 )),
  NULL)
```

`action_mature` *Gadget3 maturity action*

Description

Add maturity actions to a g3 model

Usage

```
g3a_mature_continuous(
        alpha = g3_parameterized('mat.alpha', by_stock = by_stock),
        l50 = g3_parameterized('mat.l50', by_stock = by_stock),
        beta = 0,
        a50 = 0,
        bounded = TRUE,
        by_stock = TRUE)

g3a_mature_constant(alpha = NULL, l50 = NA, beta = NULL, a50 = NA, gamma = NULL,
    k50 = NA)

g3a_mature(stock, maturity_f, output_stocks, output_ratios = rep(1/length(output_stocks),
    times = length(output_stocks)), run_f = ~TRUE,
    run_at = g3_action_order$grow,
    transition_at = g3_action_order$mature)
```

Arguments

`alpha`	A formula to substitute for α.
`l50`	A formula to substitute for l_{50}. Must be defined if *alpha* is defined.
`beta`	A formula to substitute for β.
`a50`	A formula to substitute for a_{50}. Must be defined if *beta* is defined.
`gamma`	A formula to substitute for γ.
`k50`	A formula to substitute for k_{50}. Must be defined if *gamma* is defined.
`bounded`	Should the maturity ratio be bounded to 0..1? Set TRUE if maturity is producing negative numbers of individuals.
`by_stock`	Change the default parameterisation (e.g. to be by 'species'), see `g3_parameterized`.
`stock`	`g3_stock` to mature.
`maturity_f`	A maturity formula, as defined by `g3a_mature_constant`.
`output_stocks`	List of `g3_stock`s that maturing *stock* should move into.
`output_ratios`	Vector of proportions for how to distribute into *output_stocks*, summing to 1, default evenly spread.
`run_f`	formula specifying a condition for running this action, default always runs.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.
`transition_at`	Integer order that transition actions will be run within model, see `g3_action_order`.

Details

Generally you would use g3a_growmature, which does both growth and maturity at the same time.

A model can have any number of g3a_mature actions, so long as the calling arguments are different. For instance, run_f = ~age == 5 and run_f = ~age == 7.

Value

g3a_mature_continuous: A formula object representing

$$m_0 * (\alpha \Delta L + \beta \Delta t)^\top$$

m_0 The g3a_mature_constant formula, as defined below, using parameters supplied to g3a_mature_continuous

ΔL Vector of all possible changes in length, as per current growth matrix (see g3a_grow_impl_bbinom)

Δt Length of current step as a proportion of the year, e.g. 0.25. See cur_step_size in g3a_time

g3a_mature_constant: A formula object with the following equation

$$\frac{1}{1 + e^{-\alpha(l-l_{50})-\beta(a-a_{50})-\gamma(k-k_{50})}}$$

l length of stock

l_{50} length of stock when 50% are mature

a age of stock

a_{50} age of stock when 50% are mature

k weight of stock

k_{50} weight of stock when 50% are mature

g3a_mature: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Move any maturing individuals into temporary storage, stock__transitioning_num / stock__transitioning_wgt
2. Move the contents of the temporary storage into *output_stocks*

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockmature, g3a_growmature, g3_stock

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4))
ling_mat <- g3_stock('ling_mat', seq(20, 156, 4))

# Maturity for immature ling
maturity_action <- g3a_mature(ling_imm,
    maturity_f = g3a_mature_continuous(),
    output_stocks = list(ling_mat))
```

`action_migrate` *Gadget3 migration action*

Description

Add migration to a g3 model

Usage

```
g3a_migrate_normalize(row_total = 1)

g3a_migrate(stock, migrate_f, normalize_f = g3a_migrate_normalize(),
            run_f = TRUE,
            run_at = g3_action_order$migrate)
```

Arguments

`row_total`	When calculating the proportion of individuals that will stay in place, use this total for what rows are expected to sum to.
`stock`	The `g3_stock` that will migrate in this action.
`migrate_f`	A formula describing the migration in terms of (source) `area` and `dest_area`.
`normalize_f`	Function to normalize a vector of possible destinations, to make sure fish aren't added or destroyed.
`run_f`	formula specifying a condition for running this action, default always runs.
`run_at`	Integer order that spawning actions will be run within model, see `g3_action_order`.

Details

To restrict movement to a particular step in a year, or a particular area, use *run_f*. For example:

`cur_step == 1` Migration will happen on first step of every year

`cur_step == 1 && cur_year >= 1990` Migration will happen on first step of every year after 1990

`cur_step == 2 && area = 1` Migration will happen on second step of every year, in the first area

Multiple migration actions can be added, for a separate spring and autumn migration, for instance.

The action will define the following stock instance variables for each given *stock*:

stock__migratematrix $a \times a$ array, containing proportion of (stock) moved from one area to another. If NaN, no movement has occurred

Value

g3a_migrate_normalize: A formula transforming `stock__migratematrix[,stock__area_idx]` (i.e. all possible destinations from a given area) by:

1. Squaring so values are all positive
2. Altering the proportion of static individuals so a row sums to *row_total*
3. Dividing by row_total so a row sums to 1

g3a_migrate: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Fill in *stock__migratematrix* using *migrate_f* and *normalize_f*
2. Apply movement to *stock*

See Also

g3_stock

Examples

```
areas <- list(a=1, b=2, c=3, d=4)

# NB: stock doesn't live in b, so won't figure in stock_acd__migratematrix
stock_acd <- (g3_stock('stock_acd', seq(10, 40, 10))
    %>% g3s_livesonareas(areas[c('a', 'c', 'd')]))

movement_action <- list(
    g3a_migrate(
        stock_acd,
        # In spring, individuals in area 'a' will migrate to 'd'.
        ~if (area == area_a && dest_area == area_d) 0.8 else 0,
        run_f = ~cur_step == 2),
    g3a_migrate(
        stock_acd,
        # In autumn, individuals in all areas will migrate to 'a'
        ~if (dest_area == area_a) 0.8 else 0,
        run_f = ~cur_step == 4),
    list())
```

---

action_naturalmortality

Gadget3 natural mortality action

Description

Add natural mortality to a g3 model

Usage

```
g3a_naturalmortality_exp(
        param_f = g3_parameterized('M', by_stock = by_stock, by_age = TRUE),
        by_stock = TRUE,
        action_step_size_f = ~cur_step_size)

g3a_naturalmortality(
        stock,
        mortality_f = g3a_naturalmortality_exp(),
        run_f = TRUE,
        run_at = g3_action_order$naturalmortality)
```

Arguments

`param_f`	A formula to substitute for m.
`action_step_size_f`	How much model time passes in between runs of action? defaults to `~cur_step_size`, i.e. every step. Use `action_step_size_f = 1` if action only runs yearly.
`by_stock`	Change the default parameterisation (e.g. to be by 'species'), see `g3_parameterized`.
`stock`	`g3_stock` mortality applies to.
`mortality_f`	A mortality formula, as defined by `g3a_naturalmortality_exp`.
`run_f`	formula specifying a condition for running this action, default always runs.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.

Details

A model can have any number of `g3a_naturalmortality` actions, so long as the calling arguments are different. For instance, `run_f = ~age == 5` and `run_f = ~age == 7`.

Value

g3a_naturalmortality_exp: A formula object with the following equation

$$e^{-m\Delta t}$$

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

g3a_naturalmortality: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Remove a proportion of each stock group as calculated by the mortality formula `mortality_f`

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stocknatmort, `g3a_growmature`, `g3_stock`

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)

# Natural mortality for immature ling
naturalmortality_action <- g3a_naturalmortality(ling_imm)

# NB: M is used in both g3a_naturalmortality and g3a_renewal_initabund, to
# customise, you need to make sure the definitions are in sync, for example:

M <- g3_parameterized('M', by_stock = TRUE, by_age = FALSE)
actions <- list(
    g3a_naturalmortality(ling_imm,
        g3a_naturalmortality_exp(M)),
    g3a_initialconditions_normalparam(ling_imm,
        factor_f = g3a_renewal_initabund(M = M)),
    NULL)
```

---

action_order *Standard gadget3 order of actions*

Description

Constant defining standard order of actions

Usage

```
g3_action_order
```

Details

All gadget3 actions have a *run_at* parameter. This decides the point in the model that the action will happen relative to others.

The defaults for these are set via `g3_action_order`.

Value

A named integer list

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-order.html

Examples

```
# The default action order
unlist(g3_action_order)

# View single value
g3_action_order$age
```

`action_predate` *Gadget3 predation actions*

Description

Add predation to a g3 model

Usage

```
g3a_predate_catchability_totalfleet(E)

g3a_predate_catchability_numberfleet(E)

g3a_predate_catchability_linearfleet(E)

g3a_predate_catchability_project(
    quota_f = NULL,
    landings_f = NULL,
    interim_f = g3_parameterized("quota.interim", value = 0,
        by_predator = TRUE, by_step = TRUE),
    quota_prop = g3_parameterized("quota.prop", by_predator = TRUE, value = 1),
   cons_step = g3_parameterized("cons.step", by_predator = TRUE, by_step = TRUE,
        value = quote( step_lengths / 12.0 )),
    interim_unit = unit,
    unit = c("biomass", "biomass-year", "harvest-rate", "harvest-rate-year",
             "individuals", "individuals-year") )

g3a_predate_catchability_effortfleet(catchability_fs, E)

g3a_predate_catchability_quotafleet(quota_table, E,
                                    sum_stocks = list(),
                                    recalc_f = NULL)

g3a_predate_maxconsumption(
        m0 = g3_parameterized('consumption.m0', value = 1,
            by_predator = TRUE, optimise = FALSE),
        m1 = g3_parameterized('consumption.m1', value = 0,
            by_predator = TRUE, optimise = FALSE),
        m2 = g3_parameterized('consumption.m2', value = 0,
```

```
            by_predator = TRUE, optimise = FALSE),
        m3 = g3_parameterized('consumption.m3', value = 0,
            by_predator = TRUE, optimise = FALSE),
        temperature = 0 )

g3a_predate_catchability_predator(
        prey_preferences = 1,
        energycontent = g3_parameterized('energycontent', value = 1,
            by_stock = by_stock, optimise = FALSE),
        half_feeding_f = g3_parameterized('halffeeding',
            by_predator = by_predator, optimise = FALSE),
       max_consumption = g3a_predate_maxconsumption(temperature = temperature),
        temperature = 0,
        by_predator = TRUE,
        by_stock = TRUE )

g3a_predate(
        predstock,
        prey_stocks,
        suitabilities,
        catchability_f,
        overconsumption_f = quote( dif_pmin(stock__consratio, 0.95, 1e3) ),
        run_f = ~TRUE,
        run_at = g3_action_order$predate )

# NB: Deprecated interface, use g3a_predate()
g3a_predate_fleet(fleet_stock, prey_stocks, suitabilities, catchability_f,
    overconsumption_f = quote( dif_pmin(stock__consratio, 0.95, 1e3) ),
    run_f = ~TRUE, run_at = g3_action_order$predate)

# NB: Deprecated interface, use g3a_predate() with g3a_predate_catchability_totalfleet
g3a_predate_totalfleet(fleet_stock, prey_stocks, suitabilities, amount_f,
    overconsumption_f = quote( dif_pmin(stock__consratio, 0.95, 1e3) ),
    run_f = ~TRUE, run_at = g3_action_order$predate)
```

Arguments

predstock, fleet_stock	g3_stock that describes the harvesting predators/fleet, or a list of g3_stock objects, in which case g3a_predate will be run for each in turn.
prey_stocks	List of g3_stocks that maturing *stock* should move into.
suitabilities	Either a list of stock names to formula objects, with an optional unnamed default option, or a formula object (which is always used). Each formula should define suitability of a stock, for example by using g3_suitability_exponentiall5
catchability_f	A list of formulas generated by one of the g3a_predate_catchability_* functions, which define the total biomass a fleet is able to catch.
E, landings_f	A formula defining total catch a fleet can harvest at the current time/area (totalfleet/numberfleet), or a scaling factor used to define the stock caught (lin-

earfleet/effortfleet/quotafleet).

`quota_f` A per-year quota for use during projection time periods, or for the whole model if *landings_f* is `NULL`. Generally, this will be produced by `g3_quota`.

`interim_f` A formula used to determine catch in the gap between landings data & a quota calculated from projections.

`quota_prop` A quota can apply to multiple fleets, in which case use this parameter to assign the proportion of quota available to the current fleet. Note that ideally all *quota_prop* values sum to 1, but this doesn't have to be the case (e.g. over/under-utilisation of quotas).

`cons_step` The proportion of the per-year quota that is used in each step. As with *quota_prop* ideally all values sum to 1.

`unit`, `interim_unit`
The unit that *landings_f* / *interim_f* is provided in. "biomass", "effort", "individuals" are equivalent to totalfleet / linearfleet & numberfleet respectively. (the default quota functions supply their own unit via. an attribute).

`catchability_fs`
Either a list of stock names to formula objects, with an optional unnamed default option, or a formula object (which is always used).

`quota_table` A data.frame with 'biomass' and 'quota' columns, 'biomass' a numeric column, an upper bound for total biomass amount, the final value always being `Inf`. 'quota' being a list of `formula`s, defining the quota for each, e.g. with `g3_parameterized`.

`sum_stocks` Either a list of g3_stock objects to sum when choosing a value from *quote_table*, or `NULL`, in which case choose the quota based on the current prey.

`recalc_f` A formula denoting when to recalculate the current quota. For example `~cur_step == 1` will ensure the quota is only recalculated at the beginning of the year.

`amount_f` Equivalent to E passed to g3a_predate_catchability_totalfleet.

`prey_preferences`
Either 1, indicating a Type II functional response, or >1 for a Type III functional response. Either a list of stock names to numbers, with an optional unnamed default option, or a single number to be used for all stocks.

`energycontent` A formula object for the energy content of the current prey, in in kilojoules per kilogram.

`half_feeding_f` The biomass of prey required to allow the predator to consume prey at half the maximum consumption level.

`max_consumption`
A formula for maximum consumption of the predator, in kilojoules per month. Generally generated by `g3a_predate_maxconsumption`

`m0`, `m1`, `m2`, `m3` Parameters for maximum possible consumption formula, see below.

`temperature` A formula object for the current temperature, probably generated by `g3_timeareadata`.

`overconsumption_f`
Overconsumption rule, a formula that should cap all values in *stock__consratio* to <= 95

`by_stock` Change the default parameterisation (e.g. to be by 'species'), see `g3_parameterized`.

by_predator	Change the default parameterisation (e.g. to be by 'species'), see g3_parameterized.
run_f	formula specifying a condition for running this action, default always runs.
run_at	Integer order that actions will be run within model, see g3_action_order.

Details

g3a_predate will, given a g3_fleet "predator" and a set of g3_stock preys, add predation into a model. The behaviour is driven by 2 parameters:

suitabilities Defines a predator's preference within a prey stock, normally one of the suitability functions, e.g. g3_suitability_exponentiall50

catchability_f Defines a predator's overall requirements, set with one of the catchability functions, e.g. g3a_predate_catchability_totalfleet

For the definition of each catchability function, see the values section below.

Details for custom actions: The actions will define the following stock instance variables for each given *fleet_stock* and *prey_stock*:

(predstock)__totalsuit Total suitable prey for (predstock), i.e. $\sum_{preys}^{p} \sum_{lengths}^{l} F_{pl}$

prey_stock__suit_fleet_stock Suitability of (prey_stock) for (fleet_stock), i.e. F_{pl}

(predstock)_(prey_stock)__cons Biomass of (prey_stock) caught by (predstock), by predator & prey dimensions

prey_stock__totalpredate Biomass of total consumed (prey_stock), in a prey array

prey_stock__consratio Ratio of prey_stock__totalpredate / (current biomass), capped by *over-consumption_f*

In addition, g3a_predate_fleet will generate *prey_stock__predby_predstock*, Biomass of (prey_stock) caught by (fleet_stock), in a prey array, for compatibility with older models. It is otherwise identical to g3a_predate.

A model can have any number of g3a_predate_* actions, so long as the calling arguments are different. For instance, run_f = ~age == 5 and run_f = ~age == 7.

Value

g3a_predate_catchability_totalfleet: formula objects that define a fleet's desired catch by total biomass (e.g. landings data):

$$F_{pl} = SN_{pl}W_{pl}$$

$$C_{pl} = \frac{EF_{pl}}{\sum_{preys}^{p} \sum_{lengths}^{l} F_{pl}}$$

S Suitability form *suitabilities* argument

E *E* argument, biomass caught by fleet. Generally a g3_timeareadata table containing landings data, with year/step/area/weight columns

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

g3a_predate_catchability_numberfleet: formula objects that define a fleet's desired catch by total number of stock landed (individuals, not biomass):

$$F_{pl} = SN_{pl}$$

$$C_{pl} = \frac{EF_{pl}W_{pl}}{\sum_{preys}^{p}\sum_{lengths}^{l} F_{pl}}$$

S Suitability form *suitabilities* argument

E E argument, numbers caught by fleet. Generally a `g3_timeareadata` table containing landings data, or a constant quota

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

g3a_predate_catchability_linearfleet: formula objects that define a linear relationship between desired catch and available biomass:

$$F_{pl} = SN_{pl}W_{pl}$$

$$C_{pl} = E\Delta tF_{pl}$$

S Suitability form *suitabilities* argument

E E argument, scaling factor for the stock that is to be caught, per month

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

g3a_predate_catchability_project: The formula used depends on the *unit* parameter. "biomass", "effort", "individuals" are equivalent to totalfleet / linearfleet & numberfleet respectively. However, E will switch from landings to quota once the model is projecting.

g3a_predate_catchability_effortfleet: This is a multi-species extension to linearfleet, allowing differently-parameterized catchability per-stock:

$$F_{pl} = SN_{pl}W_{pl}$$

$$C_{pl} = c_sE\Delta tF_{pl}$$

S Suitability form *suitabilities* argument

c_s *catchability_fs* argument for the current stock

E E argument, scaling factor for the stock that is to be caught, per month

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

g3a_predate_catchability_quotafleet: A formula onject that defines catch based on the available biomass of the stock multiplied by a scaling factor set according to a simple harvest control rule:

$$F_{pl} = SN_{pl}W_{pl}$$

$$C_{pl} = qE\Delta tF_{pl}$$

q quota selected from *quota_table*, corresponding to the total biomass of *sum_stocks*. For example, given `data.frame(biomass = c(10000, Inf), quota = I(list(g3_parameterized('quota.low'), g3_parameterized('quota.high'))))`, 'quota.low' will be chosen when total biomass is less than 10000, otherwise 'quota.high' will be used.

S Suitability form *suitabilities* argument

E E argument, scaling factor for the stock that is to be caught, per month

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

...if *recalc_f* is set, this will only be recaculated when true. Any other step will use the previous value.

g3a_predate_maxconsumption: formula objects that define a predator's maximum consumption:

$$M_L = m_0 \Delta t e^{(m_1 T - m_2 T^3)} L^{m_3}$$

m_x *mx* parameter, for M_L, maximum possible consumption for the predator on the current timestep

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

T *temperature* parameter, formula representing current temperature

g3a_predate_catchability_predator: formula objects that define the predator/prey relationship:

$$F_{pl} = (S E_p N_{pl} W_{pl})^{d_p}$$

$$C_{pl} = \frac{N_L M_L \psi_L F_{pl}}{E_p \sum_{preys}^{p} \sum_{lengths}^{l} F_{pl}}$$

$$\psi_L = \frac{\sum_{preys}^{p} \sum_{lengths}^{l} F_{pl}}{H \Delta t + \sum_{preys}^{p} \sum_{lengths}^{l} F_{pl}}$$

S Suitability form *suitabilities* argument

Δt Length of current step as a proportion of the year, e.g. 0.25. See `cur_step_size` in `g3a_time`

N_{pl} Number of prey in length cell for prey p, length l

W_{pl} Mean weight of prey in length cell for prey p, length l

M_L Maximum possible consumption for the predator on the current timestep, in in kilojoules per month. See `g3a_predate_maxconsumption`

L Length of the current predator

E_p *energycontent* parameter, the energy content of prey

H *half_feeding_f* parameter, the biomass of prey required to allow the predator to consume prey at half the maximum consumption level

T *temperature* parameter, formula representing current temperature

g3a_predate: An action (i.e. list of formula objects) that will...

1. For each prey, collect all suitable stock into a *predstock_prey_stock__suit* variable, using the *catchability_f* $F_p l$ formula. The units here will depend on the *catchability_f* method used.
2. After all predator consumption is done, scale consumption using the *catchability_f* $C_p l$ formula into *predstock_prey_stock__cons*, summed into *prey_stock__totalpredate*
3. Calculate *prey_stock__consratio* (ratio of consumed to available), capping using *overconsumption_f*. Update *prey_stock__num*
4. Recalculate *predstock_prey_stock__cons*, *predstock_prey_stock__suit*, post-overconsumption

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockpredator, g3_stock

Examples

```
areas <- c(a = 1, b = 2)
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock(c(species = 'ling', 'mat'), seq(20, 156, 4)) %>% g3s_age(5, 15)
lln <- g3_fleet('lln') %>% g3s_livesonareas(areas[c('a', 'b')])

# Invent a lln_landings table
lln_landings <- expand.grid(
    year = 1999:2000,
    step = c(1, 2),
    area = areas[c('a', 'b')])
lln_landings$total_weight <- floor(runif(nrow(lln_landings), min=100, max=999))

# g3a_predate_catchability_totalfleet(): Set catch accordings to landings data
predate_action <- g3a_predate_fleet(
    lln,
    list(ling_imm, ling_mat),
    suitabilities = g3_suitability_exponentiall50(by_stock = 'species'),
    catchability_f = g3a_predate_catchability_totalfleet(
        g3_timeareadata('lln_landings', lln_landings, "total_weight") ))

# g3a_predate_catchability_numberfleet(): Fixed quota of 1000 fish
predate_action <- g3a_predate_fleet(
    lln,
    list(ling_imm, ling_mat),
    suitabilities = g3_suitability_exponentiall50(by_stock = 'species'),
    catchability_f = g3a_predate_catchability_numberfleet(
        g3_parameterized(
            'quota',
            value = 1000,
            by_predator = TRUE,
            scale = 0.5,
            optimise = FALSE) ))
attr(suppressWarnings(g3_to_r(list(predate_action))), 'parameter_template')
```

`action_renewal` *Gadget3 renewal actions*

Description

Add renewal / initialconditions to a g3 model

Usage

```
g3a_renewal_vonb_recl(
        Linf = g3_parameterized('Linf', value = 1, by_stock = by_stock),
        K = g3_parameterized('K', value = 1, by_stock = by_stock),
        recl = g3_parameterized('recl', by_stock = by_stock),
       recage = g3_parameterized('recage', by_stock = FALSE, optimise = FALSE),
        by_stock = TRUE)

g3a_renewal_vonb_t0(
        Linf = g3_parameterized('Linf', value = 1, by_stock = by_stock),
        K = g3_parameterized('K', value = 1, by_stock = by_stock),
        t0 = g3_parameterized('t0', by_stock = by_stock),
        by_stock = TRUE)

g3a_renewal_initabund(
    scalar = g3_parameterized('init.scalar', value = 1, by_stock = by_stock),
   init = g3_parameterized('init', value = 1, by_stock = by_stock, by_age = TRUE),
    M = g3_parameterized('M', by_stock = by_stock, by_age = TRUE),
    init_F = g3_parameterized('init.F', by_stock = by_stock_f),
    recage = g3_parameterized('recage', by_stock = FALSE, optimise = FALSE),
    proportion_f = ~1,
    by_stock = TRUE,
    by_stock_f = FALSE)

############################## g3a_initialconditions

g3a_initialconditions_normalparam(
        stock,
        factor_f = g3a_renewal_initabund(by_stock = by_stock),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        stddev_f = g3_parameterized('init.sd', value = 10,
            by_stock = by_stock, by_age = by_age),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        age_offset = quote( cur_step_size ),
        by_stock = TRUE,
        by_age = FALSE,
        wgt_by_stock = TRUE,
        run_f = ~cur_time == 0L,
```

```
        run_at = g3_action_order$initial)

g3a_initialconditions_normalcv(
        stock,
        factor_f = g3a_renewal_initabund(by_stock = by_stock),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        cv_f = g3_parameterized('lencv', by_stock = by_stock, value = 0.1,
            optimise = FALSE),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        age_offset = quote( cur_step_size ),
        by_stock = TRUE,
        by_age = FALSE,
        wgt_by_stock = TRUE,
        run_f = ~cur_time == 0L,
        run_at = g3_action_order$initial)

############################## g3a_renewal

g3a_renewal_normalparam(
        stock,
        factor_f = g3_parameterized('rec',
            by_stock = by_stock,
            by_year = TRUE,
            scale = g3_parameterized(
                name = 'rec.scalar',
                by_stock = by_stock),
            ifmissing = g3_parameterized(
                name = 'rec.proj',
                optimise = FALSE,
                by_stock = by_stock )),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        stddev_f = g3_parameterized('rec.sd', value = 10, by_stock = by_stock),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        by_stock = TRUE,
        wgt_by_stock = TRUE,
        run_age = quote(stock__minage),
        run_projection = TRUE,
        run_step = 1,
        run_f = NULL,
        run_at = g3_action_order$renewal)

g3a_renewal_normalcv(
        stock,
        factor_f = g3_parameterized('rec',
            by_stock = by_stock,
            by_year = TRUE,
```

```
            scale = g3_parameterized(
                name = 'rec.scalar',
                by_stock = by_stock),
            ifmissing = g3_parameterized(
                name = 'rec.proj',
                optimise = FALSE,
                by_stock = by_stock )),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        cv_f = g3_parameterized('lencv', by_stock = by_stock, value = 0.1,
            optimise = FALSE),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        by_stock = TRUE,
        wgt_by_stock = TRUE,
        run_age = quote(stock__minage),
        run_projection = TRUE,
        run_step = 1,
        run_f = NULL,
        run_at = g3_action_order$renewal)

############################## g3a_otherfood

g3a_otherfood(
        stock,
       num_f = g3_parameterized('of_abund', by_year = TRUE, by_stock = by_stock,
            scale = g3_parameterized(
               'of_abund.step', value = 1, by_step = TRUE, by_stock = by_stock),
            ifmissing = "of_abund.proj" ),
        wgt_f = g3_parameterized('of_meanwgt', by_stock = by_stock),
        by_stock = TRUE,
        force_lengthvector = !any(grepl("__midlen$", all.vars(num_f))),
        run_f = quote( cur_time <= total_steps ),
        run_at = g3_action_order$initial)

g3a_otherfood_normalparam(
        stock,
        factor_f = g3_parameterized(
            'of_abund', by_year = TRUE, by_stock = by_stock,
            scale = g3_parameterized(
               'of_abund.step', value = 1, by_step = TRUE, by_stock = by_stock),
            ifmissing = "of_abund.proj" ),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        stddev_f = g3_parameterized('init.sd', value = 10,
            by_stock = by_stock, by_age = by_age),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        by_stock = TRUE,
        by_age = FALSE,
```

```
        wgt_by_stock = TRUE,
        run_f = quote( cur_time <= total_steps ),
        run_at = g3_action_order$initial)

g3a_otherfood_normalcv(
        stock,
        factor_f = g3_parameterized(
            'of_abund', by_year = TRUE, by_stock = by_stock,
            scale = g3_parameterized(
               'of_abund.step', value = 1, by_step = TRUE, by_stock = by_stock),
            ifmissing = "of_abund.proj" ),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        cv_f = g3_parameterized('lencv', by_stock = by_stock, value = 0.1,
            optimise = FALSE),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        by_stock = TRUE,
        by_age = FALSE,
        wgt_by_stock = TRUE,
        run_f = quote( cur_time <= total_steps ),
        run_at = g3_action_order$initial)
```

Arguments

stock	The g3_stock to apply to
num_f	formula that produces a lengthgroup vector of number of individuals for the current age/area/... length group.
wgt_f	formula that produces a lenghgroup vector of mean weight for the current age/area/... length group.
run_at	Integer order that actions will be run within model, see g3_action_order.
factor_f, mean_f, stddev_f, alpha_f, beta_f	formula substituted into normalparam calcuations, see below.
cv_f	formula substituted into normalcv calcuations, basically stddev_f = mean_f * cv_f, see below.
age_offset	Replace age with age - age_offset in *mean_f*. Used to simulate initialconditions at time "-1".
force_lengthvector	Should we assume that num_f is a constant, and needs repeating enough times to turn into a length vector?
run_age	Age to run renewals for, used as age == (run_age) into default *run_f*
run_projection	Boolean. Run renewal in projection years? If false adds !cur_year_projection into default *run_f*
run_step	Which step to perform renewal in, or NULL for continuous renewal. Adds cur_step == (run_step) into default *run_f*

`run_f` formula specifying a condition for running this action, For initialconditions defaults to first timestep. For renewal, the default is a combination of *run_age*, *run_step* & *run_projection*. For otherfood, the default is to always run, apart from when the model is ending.

`Linf`, `K`, `t0`, `recl` formula substituted into vonb calcuations, see below.

`recage` formula substituted into initial abundance and vonb calcuations, see below.

`proportion_f`, `scalar`, `init`, `M`, `init_F`
formula substituted into initial abundance calcuations, see below.

`by_stock`, `wgt_by_stock`, `by_stock_f`, `by_age`
Controls how parameters are grouped, see `g3_parameterized`

Details

All of the following actions will renew stock in a model. The differences are when and what they apply to by default:

`g3a_initialconditions_*` Will run at the start of the model, building an inital state of all ages

`g3a_renewal_*` Will run at every step but only for the minimal age, adding new recruits as an alternative to `g3a_spawn()`

`g3a_otherfood_*` Will run at every step, replacing the previous state, creating a non-dynamic stock for predators to consume

Specifying the quantities and mean-weights in each case works identically.

A model can have any number of `g3a_renewal_*` actions, so long as the calling arguments are different. For instance, `run_f = ~age == 5` and `run_f = ~age == 7`.

The `g3a_renewal_*` actions will define the following stock instance variables for *stock*:

stock__renewalnum Extra individuals added to the stock

stock__renewalwgt Mean weight of added individuals

Value

g3a_renewal_vonb_recl: A formula object representing

$$L_\infty(1 - e^{-\kappa(a-a_0+\frac{\log(1-L_0/L_\infty)}{\kappa})})$$

L_∞ *Linf* argument, by default model parameter named `(stock).Linf`

κ *K* argument, by default model parameter named `(stock).K`

L_0 *recl* argument, by default model parameter named `(stock).recl`

a_0 *recage* argument, by default model parameter named `recage`

NB: `g3a_initialconditions_normalparam` will replace a with $a - \Delta t$, see *age_offset*

g3a_renewal_vonb_t0: A formula object representing

$$L_\infty(1 - e^{-\kappa(a-t_0)})$$

L_∞ *Linf* argument, by default model parameter named `(stock).Linf`

κ *K* argument, by default model parameter named `(stock).K`
t_0 *t0* argument, by default model parameter named `(stock).t0`
NB: `g3a_initialconditions_normalparam` will replace a with $a - \Delta t$, see *age_offset*

g3a_renewal_vonb: An alias for `g3a_renewal_vonb_recl()`

g3a_renewal_initabund: A formula object representing

$$ps_0 s_a e^{-1(M+F_0)(a-a_0)}$$

s_0 *scalar* argument, by default model parameter named `(stock).init.scalar`
s_a *init* argument, by default model parameter named `(stock).init.(age)`
M *M* argument, by default model parameter named `(stock).M.(age)`
F_0 *init_F* argument, by default model parameter named `init.F`
a_0 *recage* argument, by default model parameter named `recage`
p *proportion* argument, by default `1`

g3a_otherfood: An action (i.e. list of formula objects) that will, for the given *stock*, iterate over each area/age/etc. combination, and generate a lengthgroup vector of new individuals and weights using *num_f* and *wgt_f*.
renewal will add fish to the existing stock, whereas initialconditions & otherfood will replace any previous values.

g3a_initialconditions_normalparam / g3a_renewal_normalparam / g3a_otherfood_normalparam: An action (i.e. list of formula objects) that will, for the given *stock*, iterate over each area/age/etc. combination, and generate new individuals.
The following formulas are used to calculate abundance (N) and mean weight (W):

$$n = dnorm(L, \mu, \sigma)$$
$$N = F10000\frac{n}{\sum n}$$
$$W = \alpha L^{\beta}$$

L Midlength of all length groups for current area/age/...
F *factor_f* argument, by default output of `g3a_renewal_initabund`
μ *mean_f* argument, by default output of `g3a_renewal_vonb_t0`
σ *stddev_f* argument, by default model parameter named `(stock).(init|rec).sd`
α *alpha_f* argument, by default model parameter named `(stock).walpha`
β *beta_f* argument, by default model parameter named `(stock).wbeta`

g3a_initialconditions_normalcv / g3a_renewal_normalcv / g3a_otherfood_normalcv: An action (i.e. list of formula objects) that will, for the given *stock*, iterate over each area/age/etc. combination, and generate new individuals.
The following formulas are used to calculate abundance (N) and mean weight (W):

$$n = dnorm(L, \mu, \mu * CV)$$
$$N = F10000\frac{n}{\sum n}$$
$$W = \alpha L^{\beta}$$

L Midlength of all length groups for current area/age/...

F *factor_f* argument, by default output of g3a_renewal_initabund

μ *mean_f* argument, by default output of g3a_renewal_vonb_t0

CV *cv_f* argument, by default model parameter named `(stock).lencv`

α *alpha_f* argument, by default model parameter named `(stock).walpha`

β *beta_f* argument, by default model parameter named `(stock).wbeta`

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockinitial,
https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec:stockrenew,
https://gadget-framework.github.io/gadget2/userguide/chap-other.html#chap-other,
g3_stock

Examples

```
stocks <- list(
    imm = g3_stock(c('ling', maturity = "imm"), seq(10, 100, 10)) %>% g3s_age(3, 7),
    mat = g3_stock(c('ling', maturity = "mat"), seq(10, 100, 10)) %>% g3s_age(5, 10) )

actions <- list(
    g3a_time(2000, 2000),
    g3a_initialconditions_normalcv(stocks$imm),
    g3a_initialconditions_normalcv(stocks$mat),
    NULL)
model_fn <- g3_to_r(c(actions, list(
    g3a_report_detail(actions) )))
attr(model_fn, 'parameter_template') |>
    g3_init_val("init.F", 0.4) |>
    g3_init_val("ling_imm.Linf", 80) |>
    g3_init_val("ling_mat.Linf", 160) |>
    g3_init_val("ling_*.K", 90) |>
    g3_init_val("ling_*.t0", 0) |>
    g3_init_val("ling_*.lencv", 0.1) |>
    g3_init_val("ling_imm.init.#", 3:7 * 100) |>
    g3_init_val("ling_mat.init.#", 5:10 * 200) |>
    g3_init_val("ling_*.init.scalar", 200) |>
    g3_init_val("ling_*.walpha", 2.275e-06) |>
    g3_init_val("ling_*.wbeta", 3.2020) |>
    g3_init_val("ling_*.M.#", 0.15) |>
    identity() -> params.in
r <- model_fn(params.in)

g3_array_plot(attr(r, "dstart_ling_imm__num")[,,time=1])
g3_array_plot(attr(r, "dstart_ling_mat__num")[,,time=1])

## Plots
par(mar = c(4,2,2,1), cex.main = 1)
curve(g3_eval(g3a_renewal_vonb_t0(Linf = 20, K = 0.8, t0 = 0), age = x),
   0, 10, col = 2, xlab = "age", main = "g3a_renewal_vonb_t0(Linf = 20, K = 0.8..1.4, t0 = 0)")
curve(g3_eval(g3a_renewal_vonb_t0(Linf = 20, K = 1.0, t0 = 0), age = x),
```

```
    0, 10, col = 1, add = TRUE)
curve(g3_eval(g3a_renewal_vonb_t0(Linf = 20, K = 1.2, t0 = 0), age = x),
    0, 10, col = 3, add = TRUE)
curve(g3_eval(g3a_renewal_vonb_t0(Linf = 20, K = 1.4, t0 = 0), age = x),
    0, 10, col = 4, add = TRUE)

## Otherfood
# "Otherfood" stocks are defined in a similar manner to any other stock
# Note that _normalparam & _normalcv need both length & age dimensions
other_wgt <- g3_stock('other_wgt', 0)
other_cv <- g3_stock('other_cv', seq(50, 100, by = 10)) %>% g3s_age(5,10)

actions <- list(
    g3a_time(2000, 2010),
    # Will get other_wgt.of_abund.1998.1, other_wgt.of_meanwgt parameters
    g3a_otherfood(other_wgt),
    # Use standard vonB parameters (Linf/K/t0) to define abundance
    g3a_otherfood_normalcv(other_cv),
    NULL)
model_fn <- g3_to_r(c(actions, list(
    g3a_report_detail(actions) )))
attr(model_fn, 'parameter_template') |>
    g3_init_val("other_cv.Linf", 80) |>
    g3_init_val("other_cv.K", 90) |>
    g3_init_val("other_cv.t0", 0) |>
    g3_init_val("other_cv.of_abund.#", 100:110) |>
    g3_init_val("other_wgt.of_abund.#", 100:110) |>
    g3_init_val("other_wgt.of_abund.step.#", 1) |>
    g3_init_val("other_cv.of_abund.proj", 80) |>
    g3_init_val("other_wgt.of_abund.proj", 70) |>
    g3_init_val("project_years", 5) |>
    identity() -> params.in
r <- model_fn(params.in)

g3_array_plot(t(attr(r, "dstart_other_wgt__num")))
g3_array_plot(t(attr(r, "dstart_other_cv__num")[,age="age7",]))
```

`action_report` *Gadget3 report actions*

Description

Add report to a g3 model

Usage

```
g3a_report_stock(report_stock, input_stock, report_f,
    include_adreport = FALSE,
    run_f = TRUE,
    run_at = g3_action_order$report)
```

```
g3a_report_history(
        actions,
        var_re = "__num$|__wgt$",
        out_prefix = "hist_",
        run_f = TRUE,
        run_at = g3_action_order$report)

g3a_report_detail(actions,
    run_f = quote( g3_param('report_detail', optimise = FALSE, value = 1L,
        source = "g3a_report_detail") == 1 ),
    abundance_run_at = g3_action_order$report_early,
    run_at = g3_action_order$report)
```

Arguments

report_stock	The g3_stock to aggregate into
input_stock	The g3_stock that will be aggregated
report_f	formula specifying what to collect, for instance g3_formula(stock_ss(input_stock__num)) or g3_formula(stock_ss(input_stock__wgt)).
actions	List of actions that model will consist of.
var_re	Regular expression specifying variables to log history for.
out_prefix	Prefix to add to history report output, e.g. hist_ling_imm__num.
include_adreport	Should the aggregated value get ADREPORT'ed?
abundance_run_at	Integer order that abundance will be collected within the model. Note that by default it's collected at the start, not the end
run_f	formula specifying a condition for running this action, default always runs.
run_at	Integer order that actions will be run within model, see g3_action_order.

Details

The g3a_report_detail defines a selection of default reports from your model, using g3a_report_history:

*_surveyindices_*__params The slope/intercept as used by g3l_distribution_surveyindices_log

step_lengths

*_weight The weighting of likelihood components

nll_* Breakdown of nll for each likelihood component

dstart_*__num Abundance in numbers, at start of each model step

dstart_*__wgt Mean weight of individuals, at start of each model step

detail_*__renewalnum Numbers produced by renewal at each model step

detail_*__spawnednum Numbers produced by spawning at each model step

detail_*_*__cons Total biomass of prey consumed by predator, at each model step

`detail_*_*__suit` Total suitable biomass of prey for predator, at each model step

The reports produced by `g3a_report_history` will vary based on the provided inputs. A model can have any number of `g3a_report_*` actions, so long as the calling arguments are different. For instance, `run_f = ~age == 5` and `run_f = ~age == 7`.

Value

g3a_report_stock: An action (i.e. list of formula objects) that will...

1. Iterate over *input_stock*, collecting data into *report_stock*
2. Add the contents of *report_stock__instance_name* to the model report

g3a_report_history: An action (i.e. list of formula objects) that will store the current state of each variable found matching *var_re*.

g3a_report_detail: Uses g3a_report_history to generate detailed reports suitable for use in `g3_fit`.

See Also

`g3_stock`

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)

# Report that aggregates ages together
agg_report <- g3_stock('agg_report', c(1)) %>%
    g3s_agegroup(list(young = 1:3, old = 4:5)) %>%
    g3s_time(year = 2000:2002)
# Generate dissaggregated report by cloning the source stock, adding time
raw_report <- g3s_clone(ling_imm, 'raw_report') %>%
    g3s_time(year = 2000:2002)

actions <- list(
    g3a_age(ling_imm),
    g3a_report_stock(agg_report, ling_imm, g3_formula( stock_ss(ling_imm__num) ),
        include_adreport = TRUE),
    g3a_report_stock(raw_report, ling_imm, g3_formula( stock_ss(ling_imm__num) )))
# "raw_report__num" and "agg_report__num" will be available in the model report
# In addition, agg_report__num will be included in TMB::sdreport() output

# Report history of all "__num" and "__wgt" variables
actions <- c(actions, list(g3a_report_history(actions)))

# Report history of just "ling_imm__num"
actions <- c(actions, list(g3a_report_history(actions, "^ling_imm__num$")))

# Add a detail report suitable for g3_fit
actions <- c(actions, list(g3a_report_detail(actions)))
```

`action_spawn` *Gadget3 spawning action*

Description

Add spawning to a g3 model

Usage

```
g3a_spawn_recruitment_fecundity(
        p0 = g3_parameterized('spawn.p0', value = 1, by_stock = by_stock),
        p1 = g3_parameterized('spawn.p1', value = 1, by_stock = by_stock),
        p2 = g3_parameterized('spawn.p2', value = 1, by_stock = by_stock),
        p3 = g3_parameterized('spawn.p3', value = 1, by_stock = by_stock),
        p4 = g3_parameterized('spawn.p4', value = 1, by_stock = by_stock),
        by_stock = TRUE )

g3a_spawn_recruitment_simplessb(
        mu = g3_parameterized('spawn.mu', by_stock = by_stock),
        by_stock = TRUE )

g3a_spawn_recruitment_ricker(
        mu = g3_parameterized('spawn.mu', by_stock = by_stock),
        lambda = g3_parameterized('spawn.lambda', by_stock = by_stock),
        by_stock = TRUE )

g3a_spawn_recruitment_bevertonholt(
        mu = g3_parameterized('spawn.mu', by_stock = by_stock),
        lambda = g3_parameterized('spawn.lambda', by_stock = by_stock),
        by_stock = TRUE )

g3a_spawn_recruitment_bevertonholt_ss3(
        # Steepness parameter
        h = g3_parameterized('spawn.h', lower = 0.1, upper = 1, value = 0.5,
            by_stock = by_stock ),
        # Recruitment deviates
        R = g3_parameterized('spawn.R', by_year = TRUE, exponentiate = TRUE,
            # Unfished equilibrium recruitment
            scale = "spawn.R0",
            by_stock = by_stock),
        # Unfished equilibrium spawning biomass (corresponding to R0)
        B0 = g3_parameterized('spawn.B0', by_stock = by_stock),
        by_stock = TRUE )

g3a_spawn_recruitment_hockeystick(
        r0 = g3_parameterized('spawn.r0', by_stock = by_stock),
        blim = g3_parameterized('spawn.blim', value = 1, by_stock = by_stock),
```

```
        by_stock = TRUE )

g3a_spawn(
        stock,
        recruitment_f,
        proportion_f = 1,
        mortality_f = 0,
        weightloss_f = 0,
        weightloss_args = list(),
        output_stocks = list(),
     output_ratios = rep(1 / length(output_stocks), times = length(output_stocks)),
        mean_f = g3a_renewal_vonb_t0(by_stock = by_stock),
        stddev_f = g3_parameterized('rec.sd', value = 10, by_stock = by_stock),
        alpha_f = g3_parameterized('walpha', by_stock = wgt_by_stock),
        beta_f = g3_parameterized('wbeta', by_stock = wgt_by_stock),
        by_stock = TRUE,
        wgt_by_stock = TRUE,
        run_step = NULL,
        run_f = ~TRUE,
        run_at = g3_action_order$spawn,
        recruit_at = g3_action_order$renewal)
```

Arguments

`p0`, `p1`, `p2`, `p3`, `p4`	Substituted into `g3a_spawn_recruitment_fecundity` formula, see below.
`mu`, `lambda`, `h`, `R`, `B0`, `r0`, `blim`	Substituted into `g3a_spawn_recruitment_*` formula, see below.
`stock`	The mature `g3_stock` that will spawn in this action.
`recruitment_f`	A list of formula generated by one of the `g3a_spawn_recruitment_*` functions, containing **s** Formula run for each subset of stock **r** Final formula for calculating number of recruits for spawning action
`proportion_f`	formula generated by one of the g3_suitability_* functions, describing the proportion of stock that will spawn at this timestep.
`mortality_f`	formula generated by one of the g3_suitability_* functions, describing the proportion of spawning stock that will die during spawning.
`weightloss_f`	formula generated by one of the g3_suitability_* functions, describing the overall weight loss during spawning. DEPRECATED: Use *weightloss_args* for new models.
`weightloss_args`	list of options to pass to `g3a_weightloss`, e.g. *rel_loss* & *abs_loss*. If not empty, a weightloss action will be included as part of spawning.
`output_stocks`	List of `g3_stock`s that will be spawned into.
`output_ratios`	Vector of proportions for how to distribute into *output_stocks*, summing to 1, default evenly spread.

`mean_f`, `stddev_f`, `alpha_f`, `beta_f`
: formula substituted into stock structure calculations, see g3a_renewal_normalparam for details.

`run_step`
: Which step to perform renewal in, or `NULL` for continuous spawning. Adds `cur_step == (run_step)` into default *run_f*.

`run_f`
: formula specifying a condition for running this action, default always runs.

`run_at`
: Integer order that spawning actions will be run within model, see `g3_action_order`.

`recruit_at`
: Integer order that recruitment from spawning will be run within model, see `g3_action_order`.

`by_stock`, `wgt_by_stock`
: Controls how parameters are grouped, see `g3_parameterized`

Details

To restrict spawning to a particular step in a year, or a particular area, use *run_f*. For example:

`cur_step == 1` Spawning will happen on first step of every year

`cur_step == 1 && cur_year >= 1990` Spawning will happen on first step of every year after 1990

`cur_step == 2 && area = 1` Spawning will happen on second step of every year, in the first area

The action will define the following stock instance variables for each given *stock* and *output_stock*:

stock__spawnprop Proportion of (stock) that are spawning in this spawning event

stock__spawningnum Numbers of (stock) that are spawning in this spawning event

output_stock__spawnednum Numbers of (output_stock) that will be produced in this spawning event

Value

g3a_spawn_recruitment_fecundity: A pair of formula objects:

$$S = l^{p_1} a^{p_2} (pN_{al})^{p_3} W_{al}^{p_4}$$

$$R = p_0 S$$

N_{al} Number of parent stock

W_{al} Weight of parent stock

p Proportion of parent stock spawning, from *proportion_f*

$p_{0..4}$ Arguments provided to function

g3a_spawn_recruitment_simplessb: A pair of formula objects:

$$S = N_{al} p W_{al}$$

$$R = \mu S$$

N_{al} Number of parent stock

W_{al} Weight of parent stock

p Proportion of parent stock spawning, from *proportion_f*

μ Argument provided to function

g3a_spawn_recruitment_ricker: A pair of formula objects:

$$S = N_{al} p W_{al}$$

$$R = \mu S e^{-\lambda S}$$

N_{al} Number of parent stock
W_{al} Weight of parent stock
p Proportion of parent stock spawning, from *proportion_f*
μ Argument provided to function
λ Argument provided to function

g3a_spawn_recruitment_bevertonholt: A pair of formula objects:

$$S = N_{al} p W_{al}$$

$$R = \frac{\mu S}{\lambda + S}$$

N_{al} Number of parent stock
W_{al} Weight of parent stock
p Proportion of parent stock spawning, from *proportion_f*
μ Argument provided to function
λ Argument provided to function

g3a_spawn_recruitment_bevertonholt_ss3: A modified beverton-holt implementation from SS3 returning a pair of formula objects:

$$S = N_{al} p W_{al}$$

$$R = \frac{4 h R_0 s R}{B_0(1-h) + S(5h-1)}$$

N_{al} Number of parent stock
W_{al} Weight of parent stock
p Proportion of parent stock spawning, from *proportion_f*
h Steepness parameter, by default provided by the `srr_h` parameter
R_0 Unfished equilibrium recruitment, by default provided by the `R0` parameter
R Recruitment deviates, by default provided by the `R` parameter table
B_0 Unfished equilibrium spawning biomass (corresponding to R0), by default provided by the `B0` parameter
λ Argument provided to function

g3a_spawn_recruitment_hockeystick: A pair of formula objects:

$$S = N_{al} p W_{al}$$

$$R = R_0 \min\left(S / B_{lim}, 1\right)$$

N_{al} Number of parent stock

W_{al} Weight of parent stock

p Proportion of parent stock spawning, from *proportion_f*

R_0 Argument `r0` provided to function

B_{lim} Argument `blim` provided to function

NB: This formula is differentiable, despite using `min()` in the definition above.

g3a_spawn: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Use *proportion_f* to calculate the total parent *stock* that will spawn
2. Use *recruitment_f* to derive the total newly spawned stock
3. Apply *weightloss* and *mortality_f* to the parent *stock*

... then, at recruitment stage ...

1. Recruit evenly into *output_stocks*, using *mean_f*, *stddev_f*, *alpha_f*, *beta_f* as-per g3a_renewal_normalparam

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec-stockspawn, https://nmfs-ost.github.io/ss3-doc/SS330_User_Manual_release.html#beverton-holt, g3a_naturalmortality, g3_stock

Examples

```
ling_imm <- g3_stock(c('ling', maturity = 'imm'), seq(20, 156, 4)) |> g3s_age(0, 10)
ling_mat <- g3_stock(c('ling', maturity = 'mat'), seq(20, 156, 4)) |> g3s_age(3, 10)

actions <- list(
    g3a_time(1990, 1994, c(6, 6)),
    g3a_initialconditions_normalcv(ling_imm),
    g3a_initialconditions_normalcv(ling_mat),
    g3a_age(ling_imm),
    g3a_age(ling_mat),

    g3a_spawn(
        # Spawn from ling_mat
        ling_mat,
        # Use Ricker Recruitment Function to calculate # of recruits from total biomass
        recruitment_f = g3a_spawn_recruitment_ricker(),
        # Proportion of ling_mat spawning exponential relationship based on length
        proportion_f = g3_suitability_exponentiall50(
            alpha = g3_parameterized("spawn.prop.alpha", value = 4, scale = -1),
            l50 = g3_parameterized("spawn.prop.l50", value = 60)),
        # Proportion of ling_mat dying during spawning linear relationship to length
        mortality_f = g3_suitability_straightline(
            alpha = g3_parameterized("spawn.mort.alpha"),
            beta = g3_parameterized("spawn.mort.beta")),
     # Weight of spawning ling_imm should reduce by a fixed absolute amount (see g3a_weightloss)
     weightloss_args = list( abs_loss = g3_parameterized("spawn.weightloss", value = 0.1) ),
        # Spawn into ling_imm
        output_stocks = list(ling_imm),
        # Spawning should happen on the first step of every year
        run_f = ~cur_step==1 ),
```

```
    NULL )
model_fn <- g3_to_r(c(actions,
    g3a_report_detail(actions),
    g3a_report_history(actions, "__spawningnum$|__offspringnum$") ))

attr(model_fn, "parameter_template") |>
    # g3a_initialconditions_normalcv()
    g3_init_val("*.Linf", 50) |>
    g3_init_val("*.t0", -1.4) |>
    g3_init_val("*.walpha", 0.1) |>
    g3_init_val("*.wbeta", 1) |>
    # g3a_spawn_recruitment_ricker()
    g3_init_val("*.spawn.mu", 1e6) |>
    g3_init_val("*.spawn.lambda", 30) |>
    identity() -> params

r <- attributes(model_fn(params))
colSums(r$dstart_ling_imm__num)
colSums(r$dstart_ling_mat__wgt)
```

`action_spmodel` *Gadget3 surplus production model*

Description

A simple production model can be used in place of a set of gadget stock dynamics actions.

Usage

```
g3a_spmodel_logistic(
        r = g3_parameterized("spm_r", lower = 0.01, upper = 1, value = 0.5,
            by_stock = by_stock),
        p = g3_parameterized("spm_p", lower = 0.01, upper = 10, value = 1,
            by_stock = by_stock),
        K = g3_parameterized("spm_K", lower = 100, upper = 1e6, value = 1000,
            by_stock = by_stock),
        by_stock = TRUE)

g3a_spmodel(
        stock,
        spm_num = g3a_spmodel_logistic(),
        spm_num_init = g3_parameterized("spm_n0", by_stock = TRUE),
        spm_wgt = 1,
        run_f = TRUE,
        run_at = g3_action_order$initial)
```

Arguments

r, p, K	Parameters for the logistic model, see value section.
by_stock	Change the default parameterisation (e.g. to be by 'species'), see g3_parameterized.
stock	g3_stock object to apply the simple production model to.
spm_num	formula to calculate the relative change in abundance, one of the g3a_spmodel_* functions.
spm_num_init	Starting point for stock abundance.
spm_wgt	formula to calculate the mean weight, if "1", then abundance in numbers == total biomass.
run_f	formula specifying a condition for running this action, default always runs.
run_at	Integer order that actions will be run within model, see g3_action_order.

Details

The actions will define the following variables in your model, which could be reported with g3a_report_history:

(stock)__renewalnum Numbers added to the abundance of *stock*

Note that the input stock should not have g3s_age, if the stock was broken up by age the model would quickly not make sense.

Value

g3a_spmodel_logistic: Returns a formula for use as *spm_num*:

$$rs(1 - (\frac{s}{K})^p)$$

s

r *r* argument, by default the (stock)_spm_r model parameter

p *p* argument, by default the (stock)_spm_p model parameter

K *K* argument, by default the (stock)_spm_K model parameter

g3a_spmodel: G3 action that maintains stock abundance / mean weight according to simple production model

See Also

g3_stock

Examples

```
# NB: Stock only has one length group, 30:40. So the stocks midlen is 35
stock_a <- g3_stock(c("stock", "a"), c(30, 40), open_ended = FALSE)
stocks <- list(stock_a)
fleet_a <- g3_fleet(c('fleet', "a"))

actions <- list(
    g3a_time(2000, 2010, step_lengths = c(6,6), project_years = 0),
```

```
    g3a_spmodel(
        stock_a ),
    g3a_predate(
        fleet_a,
        stocks,
        suitabilities = 1,
        catchability_f = g3a_predate_catchability_linearfleet(
            g3_parameterized("effort", value = 1e-1, by_predator = TRUE) )),

    # NB: Dummy parameter so model will compile in TMB
    ~{nll <- nll + g3_param("x", value = 0, optimise = TRUE)} )
actions <- c(actions, list(
    # NB: Late reporting for abundance
    g3a_report_history(actions, "__num$|__wgt$", out_prefix="dend_"),
    g3a_report_detail(actions) ))
model_fn <- g3_to_r(actions)

attr(model_fn, 'parameter_template') |>
    # Surplus production model parameters
    g3_init_val("*.spm_n0", 1e4) |>
    g3_init_val("*.spm_r", 0.1) |>
    g3_init_val("*.spm_p", 0.01) |>
    g3_init_val("*.spm_K", 1e8, lower = 0, upper = 1e20) |>

    identity() -> params.in
r <- attributes(model_fn(params.in))

barplot(r$dend_stock_a__num, las = 2)
barplot(r$detail_stock_a__renewalnum, las = 2)
```

`action_tagging` *Gadget3 tag-release action*

Description

Add tag-release to a g3 model

Usage

```
g3a_predate_tagrelease(
       fleet_stock, prey_stocks, suitabilities, catchability_f,
       output_tag_f, mortality_f = 0, run_f = ~TRUE,
       run_at = g3_action_order$predate, ...)

g3a_tag_shedding(stocks, tagshed_f, run_f = ~TRUE,
       run_at = g3_action_order$straying)
```

Arguments

`fleet_stock`	Tagging fleet, see g3a_predate_fleet
`prey_stocks`	Stocks fleet harvests, see g3a_predate_fleet
`suitabilities`	See g3a_predate_fleet
`catchability_f`	See g3a_predate_fleet
`output_tag_f`	formula specifying which numeric tag (see g3s_tag) stock will be released into. Implemented with a g3_timeareadata table, e.g.
`mortality_f`	formula generated by one of the g3_suitability_* functions, describing the proportion of tagged stock that will die during tagging.
`stocks`	Stocks that will shed tags
`tagshed_f`	formula for proportion that will shed tags at this point
`run_f`	formula specifying a condition for running this action, default always runs.
`run_at`	Integer order that spawning actions will be run within model, see `g3_action_order`.
`...`	Any further options for g3a_predate_fleet

Value

g3a_predate_tagrelease: An action (i.e. list of formula objects) that will...

1. Harvest as-per g3a_predate_fleet
2. Use *mortality_f* to apply tagging mortality to harvested stock
3. Use *output_tag_f* to decide what tag should be applied to harvested stock
4. Put harvested stock back into general circulation

g3a_tag_shedding: An action (i.e. list of formula objects) that will...

1. For each *stock*, move the proportion *tagshed_f* back to the "untagged" tag

See Also

`g3a_predate_fleet`, `g3s_tag`

Examples

```
tags <- c('H1-00', 'H1-01')
tags <- structure(seq_along(tags), names = tags)

prey_a <- g3_stock('prey_a', seq(1, 10)) %>% g3s_tag(tags)
fleet_a <- g3_fleet('fleet_a')

actions <- list(
    # NB: If g3_tag() is used in the stock, initialconditions/renewal
    # will only renew into tag == 0 (i.e. untagged)
    g3a_predate_tagrelease(
        # Setup as-per g3a_predate_fleet
        fleet_a,
        list(prey_a),
```

```
        suitabilities = list(prey_a = 1),
        catchability_f = g3a_predate_catchability_numberfleet(~100),

        # Optional tag mortality suitability
        mortality_f = g3_suitability_straightline(
            g3_parameterized('mort_alpha'),
            g3_parameterized('mort_beta')),

        # Formula to decide which tag to output into, generate table
        # with one tag per year
        output_tag_f = g3_timeareadata('fleet_a_tags', data.frame(
            year = 2000:2001,
            tag = tags[c('H1-00', 'H1-01')],
            stringsAsFactors = FALSE), value_field = "tag"),

        # Experiment only happens in spring
        run_f = ~cur_step == 2),

    g3a_tag_shedding(
        list(prey_a),
        # i.e. 0.125 will loose their tag each step
        tagshed_f = log(8)))
```

`action_time` *Gadget3 timekeeping actions*

Description

Add timekeeping to a g3 model

Usage

```
g3a_time(
        start_year,
        end_year,
        step_lengths = c(12),
        final_year_steps = quote( length(step_lengths) ),
      project_years = g3_parameterized("project_years", value = 0, optimise = FALSE),
       retro_years = g3_parameterized("retro_years", value = 0, optimise = FALSE),
        run_at = g3_action_order$initial,
        run_stop_at = g3_action_order$time)
```

Arguments

`start_year`	Year model run will start.
`end_year`	After this year, model run will stop.

`step_lengths` Either an MFDB time grouping, e.g. `mfdb::mfdb_timestep_quarterly`, or a vector of step lengths which should should sum to 12, for example, `c(3,3,3,3)` for quarterly steps within a year.

`final_year_steps`
Number of steps of final year to include. Either as an integer or quoted code, in which case it will be calcuated when the model runs. For example:

`0` Model stops before the start of *end_year* (it is exclusive)

`length(step_lengths)` Model stops at the end of *end_year* (it is inclusive)

`2` Model stops at the second step of *end_year*, mid-year if *step_lengths* is quarterly

`project_years` Number of years to continue running after the "end" of the model. Must be >= `0`

Defaults to an unoptimized `project_years` parameter, set to 0 (i.e. no projection). Generally, you would change this parameter in the parameter template, rather than changing here.

`retro_years` Adjust *end_year* to finish model early. Must be >= `0` Can be used in conjunction with *project_years* to project instead.

The true end year of the model will be `end_year - retro_years + project_years`.

Defaults to an unoptimized `retro_years` parameter, set to 0. Generally, you would change this parameter in the parameter template, rather than changing here.

`run_at`, `run_stop_at`
Integer order that actions will be run within model, see `g3_action_order`. *run_at* does year variable accounting, *run_stop_at* is when the model will finish if past the final step in the model.

Details

The actions will define the following variables in your model:

cur_time Current iteration of model, starts at 0 and increments until finished

cur_step Current step within individual year

cur_step_size Proportion of year this step contains, e.g. quarterly = 3/12

cur_year Current year

cur_step_final TRUE iff this is the final step of the year

cur_year_projection TRUE iff we are currently projecting past *end_year*

total_steps Total # of iterations (including projection) before model stops

total_years Total # of years (including projection) before model stops

Value

g3a_time: An action (i.e. list of formula objects) that will...

1. Define *cur_** variables listed above
2. If we've reached the end of the model, return *nll*

Examples

```
# Run model 2000..2004, in quarterly steps
time_action <- g3a_time(
    start_year = 2000,
    end_year = 2004,
    c(3, 3, 3, 3))
```

`action_trace` *Tracing and debugging tools*

Description

Tracing and debugging tools for a G3 model

Usage

```
g3a_trace_var(
        actions,
        check_finite = TRUE,
        check_positive = FALSE,
        check_strictly_positive = FALSE,
        on_error = c("continue", "browser", "stop"),
        print_var = FALSE,
        var_re = c("__num$", "__wgt$"))

g3a_trace_timings(
        actions,
        action_re = NULL )
```

Arguments

`actions`	A list of model actions to add tracing to
`check_finite`	Boolean, notify if variable is not finite (i.e. Inf, NA, NaN)
`check_positive`	Boolean, notify if variable is < 0
`check_strictly_positive`	Boolean, notify if variable is <= 0
`on_error`	What to do when a variable fails one of the checks? NB: "browser" will not work in a TMB-compiled model.
`print_var`	Boolean, if true print the value of the variable at the point the test fails. NB: This will not work in a TMB-compiled model.
`var_re`	Regular expression(s), variable whose name matches will be traced.
`action_re`	Regular expression(s), action step IDs that match will be traced (or all if NULL)

Details

The main reason to use `g3a_trace_var` is to find out why a model is producing NaN in reports / likelihood. Adding this to your model will help pinpoint the action this originally occurs in, so you can inspect closer for incorrect settings and/or bugs.

Suggested *var_re* settings: The *var_re* parameter chooses which variables are traced, and should be tweaked to further pinpoint the problem. Generally, once an error has been found, dig into the code (e.g. by doing `edit(g3_to_r(actions))`), and see what other variables are available for tracing. Some pre-canned suggestions follow:

`c("__num$", "__wgt$")` `(i.e. default)` This will trace abundance/weight for all stocks, and a good starting point.

`^[stock_name]__(num|wgt|cons|suit|totalpredate|consratio|feedinglevel)$` This will, once `[stock_name]` is replaced with the name of your stock, dig deeper into the predation mechanisms.

`g3a_trace_timings` will report a variable, `trace_timings`, with the min/mean/max number of seconds spent computing each step in the model. You can use `g3_to_desc` to extract more descriptive names for each step.

Value

g3_trace_var: A list of actions that will report when variables stop being finite (e.g.)

g3_trace_timings: A list of actions to report a `trace_timings` variable, with how long each step is taking

See Also

`g3a_predate_catchability_project`

Examples

```
stocks <- list(
    st = g3_stock("st", 1:10 * 10) |> g3s_age(1, 5) )

actions <- list(
    g3a_time(1990, 1995, c(3,3,3,3)),
    g3a_initialconditions_normalcv(stocks$st),
    g3a_growmature(stocks$st, impl_f = gadget3::g3a_grow_impl_bbinom(
        maxlengthgroupgrowth = 2L) ),

    NULL )
model_fn <- g3_to_r(c(actions, list(
    g3a_trace_var(actions),
    g3a_trace_timings(actions),
    g3a_report_detail(actions) )))

# Configure set of working parameters
attr(model_fn, "parameter_template") |>
    g3_init_val("*.K", 0.3) |>
```

```
    g3_init_val("*.t0", 0.2) |>
    g3_init_val("*.Linf", 80) |>
    g3_init_val("*.lencv", 0.1) |>
    g3_init_val("*.walpha", 0.01) |>
    g3_init_val("*.wbeta", 3) |>
    g3_init_val("*.M.#", 0.01) |>
    identity() -> params.in
nll <- model_fn(params.in) ; r <- attributes(nll) ; nll <- as.vector(nll)

# Show timings of each step of model
r$trace_timings

# Find more informative names with g3_to_desc
as.list(g3_to_desc(actions))

# Try setting parameters to NaN and see what fails:
r <- model_fn(params.in |> g3_init_val("*.t0", NaN))
r <- model_fn(params.in |> g3_init_val("*.bbin", NaN))
```

`action_weightloss` *Gadget3 weightloss action*

Description

Add weight loss events to a g3 model

Usage

```
g3a_weightloss(
        stock,
        rel_loss = NULL,
        abs_loss = NULL,
        min_weight = 1e-7,
        apply_to_pop = quote( stock__num ),
        run_f = TRUE,
        run_step = NULL,
        run_at = g3_action_order$naturalmortality )
```

Arguments

`stock`	The `g3_stock` that will lose weight in this action.
`rel_loss`	Fractional weight loss, `0.1` will result in the stock having 90 `NULL` means no fractional weight loss will be applied.
`abs_loss`	Absolute weight loss, applied after *rel_loss*. `NULL` means no absolute weight loss will be applied.
`min_weight`	Minimum weight below which weight cannot fall further. Should be more than zero to avoid models returning NaN.

apply_to_pop	Stock instance weightloss applies to, by default applies to whole stock. Used by g3a_spawn to apply to subset that spawned.
run_step	Which step to perform renewal in, or NULL for continuous spawning. Adds cur_step == (run_step) into default *run_f*.
run_f	formula specifying a condition for running this action, default always runs.
run_at	Integer order that spawning actions will be run within model, see g3_action_order.

Value

g3a_weightloss: An action (i.e. list of formula objects) that will, for the given *stock*...

1. Apply *rel_loss* and *abs_loss* to the parent *stock*

See Also

g3a_naturalmortality, g3a_spawn, g3_stock

Examples

```
st <- g3_stock('st', 10:20) |> g3s_age(3, 5)

actions <- list(
  g3a_time(2000, 2005, step_lengths = c(3, 3, 3, 3)),
  gadget3:::g3a_initialconditions_manual(st,
      # Set initial abundance & weight based on age
      ~1e5 + 0 * st__midlen,
      ~1000 * age + 0 * st__midlen ),
  g3a_age(st),

  g3a_weightloss(st,
      # 20% of body weight should be shed in autumn
      rel_loss = g3_parameterized("rel_loss_autumn", by_stock = TRUE, value = 0.2),
      run_step = 4 ),

  g3a_weightloss(st,
      # Remove "10" from body weight, with a minimum based on length
      abs_loss = g3_parameterized("absloss_length_mw", by_stock = TRUE, value = 10),
      min_weight = g3_formula(
          wmin.a * st__midlen^wmin.b,
          wmin.a = g3_parameterized("wmin.a", by_stock = TRUE, value = 10),
          wmin.b = g3_parameterized("wmin.a", by_stock = TRUE, value = 2) )),
  NULL)

model_fn <- g3_to_r(c(actions, g3a_report_detail(actions)))
r <- model_fn(attr(model_fn, 'parameter_template'))
g3_array_agg(attr(r, "dstart_st__wgt"), c("age", "time"))

## See g3a_spawn for an example of weightloss in spawning
```

array_utils *Gadget3 array-handling utilities*

Description

Tools to make munging array reports easier

Usage

```
g3_array_agg(
        ar,
        margins = NULL,
        agg = c(
            "sum",
            "length_mean", "length_sd",
            "predator_length_mean", "predator_length_sd" ),
        opt_time_split = !("time" %in% margins || "time" %in% ...names()),
        opt_length_midlen = FALSE,
        ... )

g3_array_combine(
        arrays,
        agg = sum,
        init_val = 0 )

g3_array_plot(
        ar,
        legend = "topright" )
```

Arguments

ar	Input array, e.g. dstart_fish__num from a model report, or list of arrays
arrays	List of input arrays, can be a nested list as generated by *cons* in g3_quota_assess
margins	dimension names to include in the final array, e.g. c("age", "year") to generate a report by-year & age. If NULL, no aggregation is done
agg	Function or character. Function to use when aggregating, or name of one of the built-in functions
init_val	The initial value to use when combining arrays
opt_time_split	Boolean, should we split up "time" into separate "year" & "step" dimensions?
opt_length_midlen	Boolean, should we convert "length"
legend	Location of legend, passed to legend's *x* parameter
...	Filters to apply to any dimension, including "year" / "step" if *opt_time_split* is TRUE. e.g. length = 40, age = 5, step = 1

Details

`g3_array_agg` allows you to both filter & aggregate an array at the same time.

Specifying a filter in *...* is simplfied in comparison to a regular R subset:

1. You can give the dimensions in any order
2. Values are always interpreted, `age = 3` will be interpreted as `"age3"`, not the third age.

For particular dimensions we have extra helpers:

age Numeric ages e.g. `age = 5` are converted to "age5", as generated by gadget3

length Numeric lengths will pick a value within groups, e.g. with lengths "10:20", "20:30", `length = 15` will pick the smaller lengthgroup

`g3_array_combine` generates the union of 2 disjoint arrays, so you can combine aggregated output from an immature and mature stock for example.

`g3_array_plot` will plot the contents of an array, for arrays with 2 dimensions or less.

Value

An array, filtered by *...* and aggregated by *margins*.

If *ar* was a list, a list of filtered/aggregated arrays

Examples

```
# Generate an array to test with
dn <- list(
    length = c("50:60", "60:70", "70:Inf"),
    age = paste0("age", 0:5),
    time = paste0(rep(1990:1996, each = 2), c("-01", "-02")) )
ar <- array(
    seq_len(prod(sapply(dn, length))),
    dim = sapply(dn, length),
    dimnames = dn)
ar[,,"1994-02", drop = FALSE]
g3_array_plot(ar[,,"1994-02"])
g3_array_plot(ar["50:60","age3",])

# Generate by-year report for ages 2..4
g3_array_agg(ar, c('age', 'year'), age = 2:4)

# ...for only step 1
g3_array_agg(ar, c('age', 'year'), age = 2:4, step = 1)

# Report on smallest length group, for each timestep
g3_array_agg(ar, c('length', 'time'), length = 55)
# Use midlen as the dimension name
g3_array_agg(ar, c('length', 'time'), length = 55, opt_length_midlen = TRUE)

# Combine 2 arrays with disjoint age ranges into one list
g3_array_combine(list(
```

```
    g3_array_agg(ar, c('age', 'year'), age = 2:4),
    g3_array_agg(ar / 1000, c('age', 'year'), age = 3:5) ))

# We can aggregate lists of arrays, applying the same options for each
list(a = ar, b = ar * 10) |> g3_array_agg(c('year', 'age'), length = 55)

# We can aggregate then combine
list(a = ar, b = ar * 10) |>
     g3_array_agg(c('year', 'age'), length = 55) |> g3_array_combine()
```

`env_dif` *g3 env: differentiable functions*

Description

Differentiable helper functions available to any gadget3 model

Details

These functions are part of `g3_env` is the top-level environment that any gadget3 model uses.

dif_pmax

```
dif_pmax(a, b, scale)
```

Returns the maximum of *a* & *b*. If *a* is a vector/array, then all members of *a* are compared against *b*. If *b* is also a vector, then all members of *a* are compared against the matching member of *b* (repeating *b* if necessary).

scale influences the sharpness of inflection points, should be about 1e5, depending on ranges of input values.

dif_pmin

```
dif_pmin(a, b, scale)
```

Returns the minimum of *a* & *b*, otherwise works like `dif_pmax`.

scale influences the sharpness of inflection points, should be about 1e5, depending on ranges of input values.

dif_pminmax

```
dif_pminmax(a, lower, upper, scale)
```

Returns values of *a* bounded between *lower* & *upper*.

scale influences the sharpness of inflection points at *lower* & *upper*, should be about 1e5, depending on ranges of input values.

Examples

```
## dif_pmax
g3_eval(quote( dif_pmax(1:10, 5, 1e5) ))
g3_eval(quote( dif_pmax(1:10, c(4, 7), 1e5) ))
g3_eval(quote( dif_pmax(array(1:9, dim = c(3,3)), c(3,6,8), 1e5) ))

## dif_pmin
g3_eval(quote( dif_pmin(1:10, 5, 1e5) ))
g3_eval(quote( dif_pmin(1:10, c(4, 7), 1e5) ))

## dif_pminmax
g3_eval(quote( dif_pminmax(1:10, 3, 6, 1e5) ))
```

`eval` *Evaluate G3 forumulas*

Description

Evaluate G3 formulas / code outside a model

Usage

```
g3_eval(f, ...)
```

Arguments

`f`	A formula object or quoted code to be evaluated
`...`	Named items to add to the formula's environment, or a single list / environment to use.

Details

Allows snippets of gadget3 code to be run outside a model. This could be done with regular `eval`, however, `g3_eval` does a number of things first:

1. The global `g3_env` is in the environment, so functions such as `avoid_zero` can be used
2. If substituting a `g3_stock`, all definitions such as `stock__minlen` will also be substituted
3. `g3_param('x')` will pull `param.x` from the environment

Value

Result of evaluating f.

Examples

```
# Evaluate suitiability function for given stocks
g3_eval(
    g3_suitability_andersen(0,1,2,3,4),
    predator_length = 100,
    stock = g3_stock('prey', 1:10))

# Parameters can be filled in with "param." items in environment
g3_eval(quote( g3_param('x') ), param.x = 88)
g3_eval(
    g3_parameterized('lln.alpha', by_stock = TRUE, value = 99),
    stock = g3_stock("fish", 1:10),
    param.fish.lln.alpha = 123)

# Graph gadget3's built-in logspace_add()
if (interactive()) {
  curve(g3_eval(quote( logspace_add(a, 10) ), a = x), 0, 50)
}
```

`formula_utils` *Gadget3 formula helpers*

Description

Tools to create R formulas

Usage

```
g3_formula(code, ...)
```

Arguments

code	Unevaluated code to be turned into a formula
...	Named items to add to the formula's environment, or a single list / environment to use.

Details

When using ~, the local environment is attached to the code. This can leak unwanted variables into a model. This allows you to avoid the problem without resorting to local.

Value

A formula object, with environment created from .... Can then be used anywhere in gadget3 that accepts a formula.

Examples

```
# g3_formula is identical to defining a formula within local():
stopifnot(all.equal(
    g3_formula(x + 1, z = 44),
    local({ z = 44; ~x + 1 })
    ))

# If the code is destined for CRAN, you need to quote() to avoid check errors:
stopifnot(all.equal(
    g3_formula(quote( x + 1 ), z = 44),
    local({ z = 44; ~x + 1 })
    ))
```

---

init_val *Gadget3 parameter value setter*

Description

Helper for setting initial parameter value

Usage

```
g3_init_val(
        param_template,
        name_spec,
        value = NULL,
        spread = NULL,
       lower = if (!is.null(spread)) min(value * (1-spread), value * (1+spread)),
       upper = if (!is.null(spread)) max(value * (1-spread), value * (1+spread)),
        optimise = !is.null(lower) & !is.null(upper),
        parscale = if (is.null(lower) || is.null(upper)) NULL else 'auto',
        random = NULL,
        auto_exponentiate = TRUE)
```

Arguments

param_template	A parameter template generated by g3_to_r or g3_to_tmb
name_spec	A glob-like string to match parameter names, see Details
value	Numeric value / vector of values to set for value / 'value' column in template. Original value left if NULL
spread	Shortcut for setting *lower* & *upper*.
lower	Numeric value / vector of values to set for 'lower' column in template. Original value left if NULL
upper	Numeric value / vector of values to set for 'upper' column in template. Original value left if NULL

`optimise` Boolean value to set for 'optimise' column in template. Default is true iff both lower and upper are non-NULL. Original value left if NULL

`parscale` Numeric value / vector of values to set for 'parscale' column in template. Default (`auto`) is difference between lower & upper (or NULL if they're not set). Original value left if NULL

`random` Boolean value to set for 'random' column in template. If set to TRUE, any existing optimise/lower/upper/parscale values will be cleared. Original value left if NULL

`auto_exponentiate` If TRUE, will implicitly match parameters ending with "_exp", and if this is the case `log` all *value*/*lower*/*upper* values

Details

name_spec is a glob (or wildcard) matching parameters. It is a string separated by `.`, where each part can be:

1. A wildcard matching anything (`*`), or a matching anything with a prefix, e.g. `m*`
2. A wildcard matching any number (#), or a matching a number with a prefix, e.g. `age*`
3. A range of numbers, e.g. `[1979-1984]`
4. A choice of options can be separated with `|`, e.g. `init|rec` or `[1979-1984]|[2000-2003]`

Value

A new parameter template list/table containing modifications

See Also

`g3_parameterized`

Examples

```
# A parameter template, would already be got via. attr(g3_to_tmb(...), "parameter_template")
pt <- data.frame(
    switch = c(
        paste0('fish.init.', 1:9),
        paste0('fish.rec.', 1990:2000),
        'fish.M'),
    value = NA,
    lower = NA,
    upper = NA,
    parscale = NA,
    optimise = FALSE,
    random = FALSE)

# Set all fish.init.# parameters to optimise
pt <- g3_init_val(pt, 'fish.init.#', 4, spread = 8)

# Set a fixed value for any .M
pt <- g3_init_val(pt, '*.M', value = 0.3, optimise = FALSE)
```

```
# Set a fixed value for a range of recruitment years, optimise the rest
pt |>
    g3_init_val('*.rec.#', value = 4, lower = 0, upper = 10) |>
    g3_init_val('*.rec.[1993-1996]', value = 0, optimise = FALSE) |>
    identity() -> pt

pt
```

`language` *G3 language extensions to R*

Description

Additional meta-functions available for use in G3 formula.

Details

Whilst used as functions, these functions alter the code output of the model, rather than appearing directly.

g3_idx

Adds a `- 1` to the supplied expression, but only in C++ (which has 0-based indexes). Under R the expression is passed through unchanged.

Note: This is generally for internal use, as `[[` will do this automatically for you.

For example, `g3_idx(a)` will be replaced with `a` in R output and `a - 1` in C++ output.

g3_param

Reference a scalar parameter by name. Arguments:

name Variable name for parameter. Required

value Initial value in model parameter_template. Default 0

optimise Initial optimise setting in parameter_template. Default TRUE

random Initial random setting in parameter_template. Default FALSE

lower Initial lower setting in parameter_template. Default NA

upper Initial upper setting in parameter_template. Default NA

For example, `g3_param("ling.Linf")` will register a scalar parameter called *ling.Linf*, available in the model parameter template, and be replaced by a reference to that parameter.

`g3_param("ling.Linf")` can be used multiple times, to refer to the same value.

g3_param_vector

Reference a vector parameter by name. Arguments:

name Variable name for parameter. Required

value Initial value for use in model paramter_template. Default 0

Same as `g3_param`, but the parameter will be expected to be a vector. You can then dereference with `[[`.

For example, `g3_param_vector("lingimm.M")[[age - 3 + 1]]`.

g3_param_table

Reference a lookup-table of parameters.

name Variable name for parameter. Required

table A data.frame, one column for each variable to check, one row for possible values. Required

value Initial value(s) for use in model parameter_template. Default 0

optimise Initial optimise setting in parameter_template. Default TRUE

random Initial random setting in parameter_template. Default FALSE

lower Initial lower setting(s) in parameter_template. Default NA

upper Initial upper setting(s) in parameter_template. Default NA

ifmissing Value to return when outside of table bounds. Default NaN with warning if a value is missing

This is similar to providing a vector, but can use values in the model to provide bounds-checking.

The function takes 2 arguments, a prefix for the generated parameters, and a data.frame of variables to possible values. `expand.grid` can be used to produce a cross product of all provided variables.

value, *lower*, *upper* can be vectors, in which case it is split up with one per parameter.

Note: The variables referenced will need to be integer variables, most likely iteration variables such as `cur_year`, `age`, `area`...

For example, the following: `g3_param_table('lingimm.M', expand.grid(age = seq(ling_imm__minage, ling_imm__maxage)))` will generate parameters *lingimm.M.3*..*lingimm.M.10*, assuming that ling_imm has ages 3..10.

The call to `g3_param_table` will be replaced with `param[[paste("lingimm.M", age, sep = ".")]]`, or equivalent code in C++.

g3_with

`g3_with(var1 := val1, var2 := val2, { x <- val1 * val2 })` is equivalent to `local({var1 <- val1, var2 <- val2, { x <<- val1 * val2 } })`

However, we don't make a new environment for the code block in R, only in C++.

`likelihood_bounds_penalty` *Gadget3 likelihood bounds_penalty action*

Description

Add a liklihood penalty for parameters leaving the bounds set in parameter_template

Usage

```
g3l_bounds_penalty(
        actions_or_parameter_template,
        weight = 1,
        scale = 1e6,
        run_at = g3_action_order$likelihood)
```

Arguments

`actions_or_parameter_template`	Either: A list of actions, to extract parameters from and to add bounds to. A parameter template generated by `g3_to_tmb`, with *optimise*, *lower*, *upper* populated, bounds for the parameters will be hard-coded.
`weight`	Weighting applied to this likelihood component.
`scale`	Influences the sharpness of inflection points, smaller values mean a more gentle transition.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.

Details

Whilst lower/upper can be passed to `optim`, not all methods can use them. Adding `g3l_bounds_penalty` OTOH can be used with any method.

Value

g3l_bounds_penalty: An action (i.e. list of formula objects) that will... If a *actions* list is supplied, add a large number to likelihood when any parameter is outside bounds. Bounds are updated whenever `g3_tmb_adfun` is run.

If a *parameter_template* is supplied, add a large number to likelihood when outside the bounds in the template. The bounds are baked into the model at this point.

Examples

```
anch <- g3_stock('anch', seq(20, 156, 4)) %>% g3s_age(3, 10)
actions <- list(
  g3a_time(1990, 1994),
  g3a_growmature(anch, g3a_grow_impl_bbinom(
    maxlengthgroupgrowth = 38L)),
  g3a_naturalmortality(anch),
  g3a_initialconditions_normalparam(anch),
  g3a_renewal_normalparam(anch,
    run_step = NULL),
  g3a_age(anch),
  NULL)

# Generate code with bounds added
model_code <- g3_to_tmb(c(actions, list(g3l_bounds_penalty(actions))))

attr(model_code, "parameter_template") %>%
  # Set lower / upper bounds for initial conditions
  g3_init_val("*.init.#", 10, lower = 0.001, upper = 200) %>%
  identity() -> params.in

# The objective function produced by g3_tmb_adfun() will honour the bounds
# above, without having to pass them to stats::optim()
```

---

likelihood_catchdistribution

Gadget3 likelihood actions

Description

Gather nll in a g3 model

Usage

```
g3l_distribution_sumofsquares(
     over = c('area', 'predator', 'predator_tag', 'predator_age', 'predator_length'))

g3l_distribution_multinomial(epsilon = 10)

g3l_distribution_multivariate(rho_f, sigma_f, over = c("area"))

g3l_distribution_surveyindices_log(alpha = NULL, beta = 1)

g3l_distribution_surveyindices_linear(alpha = NULL, beta = 1)

g3l_distribution_sumofsquaredlogratios(epsilon = 10)

g3l_abundancedistribution(
```

```
        nll_name,
        obs_data,
        fleets = list(),
        stocks,
        function_f,
        predators = list(),
        transform_fs = list(),
        missing_val = 0,
        area_group = NULL,
        report = FALSE,
        nll_breakdown = FALSE,
        weight = g3_parameterized(paste0(nll_name, "_weight"),
            optimise = FALSE, value = 1),
        run_at = g3_action_order$likelihood)

g3l_catchdistribution(
        nll_name,
        obs_data,
        fleets = list(),
        stocks,
        function_f,
        predators = list(),
        transform_fs = list(),
        missing_val = 0,
        area_group = NULL,
        report = FALSE,
        nll_breakdown = FALSE,
        weight = g3_parameterized(paste0(nll_name, "_weight"),
            optimise = FALSE, value = 1),
        run_at = g3_action_order$likelihood)

g3_distribution_preview(
        obs_data,
        predators = list(),
        fleets = list(),
        stocks = list(),
        area_group = NULL)
```

Arguments

over	When comparing proportions of lengthgroups, specifies the dimensions that define the total. For example the default "area" means the proprtion of the current lengthgroup to all individuals in that area. `c('area', 'predator_tag', 'predator_age', 'predator_length'))` will compare the current lengthgroup to all individuals consumed by that predator. Note that any unknown dimensions will be ignored; for example a fleet does not have a tag/age/length, so only area will have an effect here.
rho_f, sigma_f	formula substituted into multivariate calcuations, see below.

`epsilon`	Value to be used whenever the calculated probability is very unlikely. Default 10.
`alpha`	formula substituted into surveyindices calcuations to fix intercept of linear regression, or NULL if not fixed. See below.
`beta`	formula substituted into surveyindices calcuations to fix slope of linear regression, or NULL if not fixed. See below.
`nll_name`	Character string, used to define the variable name for *obsstock* and *modelstock*.
`obs_data`	Data.frame of observation data, for example the results of mfdb_sample_count. Should at least have a year column, and a length or weight column. For more information, see "obs_data and data aggregation" below.
`fleets`, `predators`	A list of `g3_stock` objects to collect catch data for. If empty, will collect abundance data for *stocks* instead.
`stocks`	A list of `g3_stock` objects to collect catch or abundance data for, depending if *stocks* were provided.
`function_f`	A formula to compare *obsstock__x* to *modelstock__x* and generate nll, defined by one of the *g3l_distribution_** functions. This will be adapted to compare either number (*modelstock__num*) or weight (*modelstock__wgt*) depending on what columns *obs_data* has.
`transform_fs`	A list of dimension names to either formula objects or list of stock names to formula objects (where the transform differs between stocks). See examples.
`missing_val`	Where there are missing values in the incoming data, value to replace them with.
`area_group`	mfdb_group or list mapping area names used in *obs_data* to integer model areas, see "obs_data and data aggregation" below.
`report`	If TRUE, add model and observation arrays to the model report, called `cdist_`*nll_name*`_model__num/wgt` and `cdist_`*nll_name*`_obs__num/wgt` respectively
`nll_breakdown`	Should the nll report be broken down by time? `TRUE` / `FALSE`
`weight`	Weighting applied to this likelihood component. Default is a `g3_param` that defaults to 1, allowing weights to be altered without recompiling.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.

Details

The actions will define the following variables in your model:

*obsstock*__**num/wgt** A `g3_stock` instance that contains all observations in an array

*modelstock*__**num/wgt** A `g3_stock` instance that groups in an identical fashion to *obsstock*, that will be filled with the model's predicted values

The model report will contain nll_cdist_*nll_name*__num and/or nll_cdist_*nll_name*__wgt, depending on the columns in *obs_data* (a number column will compare by individuals, and produce a corresponding num report). If *nll_breakdown* is `TRUE`, this will be an array with one entry per timestep.

g3l_abundancedistribution compares abundance of stocks, *g3l_catchdistribution* compares fleet catch. Thus providing fleets is mandatory for *g3l_catchdistribution*, and an error for *g3l_abundancedistribution*.

obs_data and data aggregation: The *obs_data* data.frame, as well as providing the observation data to compare the model data against, controls the grouping of model data to compare to the observation data, by inspecting the MFDB column attributes produced by e.g. mfdb_sample_count.

Metadata columns describe the observation datapoint in that row. The columns should be from this list:

year Required. Year for the data point. Gaps in years will result in no comparison for that year

step Optional. If there is no step column, then the data is assumed to be yearly, and the model data for all timesteps will be summed before comparing.
Model timestep for the data point. Gaps in steps will result in no comparison for that year/step.

length Optional. If missing all lengthgroups in the model will be summed to compare to the data.
The column can be a factor, as generated by `cut()`, e.g `cut(raw_length, c(seq(0, 50, by = 10), Inf), right = FALSE)` for an open-ended upper group.
The column can be character strings also formatted as factors as above. The column entries are assumed to be sorted in order and converted back to a factor.
If `open_ended = c('lower', 'upper')` was used when querying MFDB for the data, then the bottom/top length groups will be modified to start from zero or be infinite respectively.
Any missing lengthgroups (when there is otherwise data for that year/step) will be compared to zero.

age Optional. If missing all age-groups (if any) in the model will be summed to compare to the data.
Model ages will be grouped by the same groupings as MFDB used, thus if the data was formed with a query `age = mfdb_group(young = 1:3, old = 4:5)`, then the model data will similarly have 2 groups in it.
Any missing ages (when there is otherwise data for that year/step) will be compared to zero.

predator_length / predator_age / predator_tag Optional.
Values are the same as with length/age/tag respectively, but group by the predator rather than the prey.

stock Optional. If tmissing all stocks in *stocks* will be summed to compare to the data.
The values in the stocks column should match the names of the stocks given in the *stocks* parameter. This column can be factor or character.
The values can also some of the stock name parts, e.g. `"st_f"` or `"f"` which would then aggregate `"st_imm_f"`, `"st_mat_f"` together.
However, note that a stock can only be included in one grouping, so given columns `"f"` & `"imm"`, `"st_imm_f"` would only be included in the former group. If you want to do something along these lines, 2 separate likelihood actions would be more appropriate.
Any missing stocks (when there is otherwise data for that year/step) will be compared to zero.

stock_re Optional. If this and stock are missing all stocks in *stocks* will be summed to compare to the data.
The values in the stocks column will be used as regular expressions to match the names of the stocks given in the *stocks* parameter. For example, '_mat_' will match both 'ghd_mat_f' and 'ghd_mat_m' and will be compared against the sum of the 2 stocks.
Any missing stocks (when there is otherwise data for that year/step) will be compared to zero.

fleet Optional. If this and fleet_re are missing all fleets in *fleets* will be summed to compare to the data.
The values in the fleets column should match the names of the fleets given in the *fleets* parameter. This column can be factor or character.

Any missing fleets (when there is otherwise data for that year/step) will be compared to zero.

fleet_re Optional. If this and fleet are missing all fleets in *fleets* will be summed to compare to the data.
The values in the fleets column will be used as regular expressions to match the names of the fleets given in the *fleets* parameter. For example, '_trawl_' will match both 'fleet_trawl_is' and 'fleet_trawl_no' and will be compared against the sum of the 2 fleets.
Any missing fleets (when there is otherwise data for that year/step) will be compared to zero.

area Optional. If missing all areas in the model will be summed to compare to the data.
Unlike other columns, the MFDB grouping here is ignored (the areas it is grouping over aren't integer model areas). Instead, the *area_group* parameter should describe how to map from the area names used in the table to integer model areas.
For example, if `area_group = list(north=1:2, south=3:5)`, then the area column of *obs_data* should contain either "north" or "south", and corresponding model data will be summed from integer model areas 1,2 and 3,4,5 respectively.
If *area_group* is not supplied, then we assume that *obs_data* area column will contain model area integers.
Any missing areas (when there is otherwise data for that year/step) will be compared to zero.

Data columns contain the observation data to compare. There should be at least one of:

number If a number column appears in *obs_data*, then the stock abundance by individuals will be aggregated and compared to the *obs_data* number column.

weight If a weight column appears in *obs_data*, then the total biomass of the stock will be aggregated and compared to the *obs_data* number column.

You can use `g3_distribution_preview` to see how your observation data will be converted into an array.

Value

g3l_distribution_sumofsquares: Returns a formula for use as *function_f*:

$$\sum_{lengths} \left(\frac{N_{tral}}{N_{tr}} - \frac{\nu_{tral}}{\nu_{tr}} \right)^2$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

N_{tr} Total observation sample size for current time/area (or dimensions set in *over*)

ν_{tr} Total model sample size for current time/area (or dimensions set in *over*)

g3l_distribution_multinomial: Returns a formula for use as *function_f*:

$$2(\sum_{lengths} \log N_{tral}! - \log(\sum_{lengths} N_{tral})! - \sum_{lengths} (N_{tral} \log min(\frac{\nu_{tral}}{\sum_{lengths} \nu_{tral}}, \frac{1}{l\epsilon})))$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

l Number of lengthgroups in sample

ϵ *epsilon* parameter

g3l_distribution_multivariate: Returns a formula for use as *function_f*, which calls TMB's `SCALE(AR1(rho), sigma)(x)`, where *rho* and *sigma* are parameters, and `x` is defined as:

$$\frac{N_{tral}}{N_{tr}} - \frac{\nu_{tral}}{\nu_{tr}}$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

N_{tr} Total observation sample size for current time/area (or dimensions set in *over*)

ν_{tr} Total model sample size for current time/area (or dimensions set in *over*)

For more information, see Autoregressive processes in the TMB book.

g3l_distribution_surveyindices_log: Returns a formula for use as *function_f*:

$$\sum_{time} (\alpha + \beta \log N_{tral} - \log \nu_{tral})^2$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

α *alpha* parameter

β *beta* parameter

If *alpha* or *beta* is not provided, then linear regression is performed on N, ν over time for each area/age/length combination. The used values will be stored in a `cdist_`*nll_name*`_model__param` array and reported after model run, whether calculated or hard-coded.

g3l_distribution_surveyindices_linear: Returns a formula for use as *function_f*:

$$\sum_{lengths} (\alpha + \beta N_{tral} - \nu_{tral})^2$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

α *alpha* parameter

β *beta* parameter

If *alpha* or *beta* is not provided, then linear regression is performed on N, ν over time for each area/age/length combination. The used values will be stored in a `cdist_`*nll_name*`_model__param` array and reported after model run, whether calculated or hard-coded.

g3l_distribution_sumofsquaredlogratios: The equivalent of gadget2's `catchinkilos`. Returns a formula for use as *function_f*:

$$\sum_{lengths} (log(N_{tral} + \epsilon) - log(\nu_{tral} + \epsilon))^2$$

N_{tral} Observation sample size for current time/area/age/length combination

ν_{tral} Model sample size for current time/area/age/length combination

ϵ *epsilon* parameter

g3l_abundancedistribution: An action (i.e. list of formula objects) that will...

1. For all *stocks*, collect catch data into *modelstock*__num or *modelstock*__wgt, depending on the columns provided in *obs_data*
2. Compare *modelstock*__num/wgt with *obsstock*__num/wgt, using *function_f*

The output of *function_f* is summed over all stock dimensions (age/area) and time and added to `nll`.

g3l_catchdistribution: An action (i.e. list of formula objects) that will...

1. For all *fleets* and *stocks* combinations, collect catch data into *modelstock*__num or *modelstock*__wgt, depending on the columns provided in *obs_data*
2. Compare *modelstock*__num/wgt with *obsstock*__num/wgt, using *function_f*

The output of *function_f* is summed over all stock dimensions (age/area) and time and added to `nll`.

g3_distribution_preview: The input *obs_data* formatted as an array, applying the same rules that `g3l_*distribution` will.

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-like.html, g3_stock

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock('ling_mat', seq(20, 156, 4)) %>% g3s_age(5, 15)
lln <- g3_fleet('lln')

# Invent a ldist.lln table for our tests
ldist.lln.raw <- data.frame(
    year = c(1999, 2000),
    age = sample(5:9, 100, replace = TRUE),
    length = sample(10:70, 100, replace = TRUE),
    number = 1,
    stringsAsFactors = FALSE)

# Group length into 10-long bins
# NB: The last 2 bins will be empty, but gadget3 will use the factor levels, include them as zero
# NB: Generally one would use mfdb::mfdb_sample_count() source and group data for you
ldist.lln.raw |> dplyr::group_by(
  year = year, age = age,
  length = cut(length, breaks = seq(10, 100, by = 10), right = FALSE)
) |> dplyr::summarise(number = sum(number), .groups = 'keep') -> ldist.lln

# Turn age into a factor, indicating all ages we should be interested in
ldist.lln$age <- factor(ldist.lln$age, levels = 5:15)

# We can see the results of this being turned into an array:
g3_distribution_preview(ldist.lln)

likelihood_actions <- list(
  g3l_catchdistribution(
    'ldist_lln',
```

```
    ldist.lln,
    fleets = list(lln),
    stocks = list(ling_imm, ling_mat),
    g3l_distribution_sumofsquares()))

# Make an (incomplete) model using the action, extract the observation array
fn <- suppressWarnings(g3_to_r(likelihood_actions))
environment(fn)$cdist_sumofsquares_ldist_lln_obs__num

# Apply age-reading error matrix to model data
more_likelihood_actions <- list(
  g3l_catchdistribution(
    'ldist_lln_readerror',
    ldist.lln,
    fleets = list(lln),
    stocks = list(ling_imm, ling_mat),
    transform_fs = list(age = g3_formula(
      g3_param_array('reader1matrix', value = diag(5))[g3_idx(preage), g3_idx(age)]
      )),
    g3l_distribution_sumofsquares()))

# Apply per-stock age-reading error matrix to model data
more_likelihood_actions <- list(
  g3l_catchdistribution(
    'ldist_lln_readerror',
    ldist.lln,
    fleets = list(lln),
    stocks = list(ling_imm, ling_mat),
    transform_fs = list(age = list(
      ling_imm = quote( g3_param_array('imm_readermatrix',
          value = diag(ling_imm__maxage - ling_imm__minage + 1)
          )[ling_imm__preage_idx, ling_imm__age_idx] ),
      ling_mat = quote( g3_param_array('mat_readermatrix',
          value = diag(ling_mat__maxage - ling_mat__minage + 1)
          )[ling_mat__preage_idx, ling_mat__age_idx] ),
      unused = 0)),
    g3l_distribution_sumofsquares()))

## Stomach content: predator-prey species preference
prey_a <- g3_stock('prey_a', seq(1, 10)) |> g3s_age(1,3)
prey_b <- g3_stock('prey_b', seq(1, 10)) |> g3s_age(1,3)
pred_a <- g3_stock('pred_a', seq(50, 80, by = 10)) |> g3s_age(0, 10)
otherfood <- g3_stock('otherfood', 0)

# Produce data.frame with columns:
# * predator_length or predator_age
# * stock
# * number or weight
pred_a_preypref_obs <- expand.grid(
    year = 2000:2005,
    predator_length = c(50,70),
    stock = c('prey_a', 'prey_b', 'otherfood'),
    number = 0 )
```

```
# Create catchdistribution likelihood component
actions <- list(
    g3l_catchdistribution(
        'pred_a_preypref',
        pred_a_preypref_obs,
        fleets = list(pred_a),
        stocks = list(prey_a, prey_b, otherfood),
        g3l_distribution_sumofsquares(),
        nll_breakdown = TRUE,
        report = TRUE ),
    NULL)

## Stomach content: predator-prey size preference
# Produce data.frame with columns:
# * predator_length or predator_age
# * (prey) length
# * number or weight
pred_a_sizepref_obs <- expand.grid(
    year = 2000:2005,
    predator_length = c(50,70),
    length = seq(1, 10),
    number = 0 )

# Create catchdistribution likelihood component
actions <- list(
    g3l_catchdistribution(
        'pred_a_sizepref',
        pred_a_sizepref_obs,
        predators = list(pred_a),
        # NB: Only referencing stocks included in observation data
        stocks = list(prey_a),
        function_f = g3l_distribution_sumofsquares(),
        # Use transform_fs to apply digestioncoefficients
        transform_fs = list(length = list(prey_a = g3_formula(
            quote( diag(d0 + d1 * prey_a__midlen^d2) ),
            d0 = g3_parameterized('d0', by_stock = TRUE),
            d1 = g3_parameterized('d1', by_stock = TRUE),
            d2 = g3_parameterized('d2', by_stock = TRUE) ))),
        nll_breakdown = TRUE,
        report = TRUE ),
    NULL)
```

`likelihood_random` *Gadget3 random effects likelihood actions*

Description

Add likelihood components for random effects

Usage

```
g3l_random_dnorm(
        nll_name,
        param_f,
        mean_f = 0,
        sigma_f = 1,
        log_f = TRUE,
        period = 'auto',
        nll_breakdown = FALSE,
        weight = g3_parameterized(paste0(nll_name, "_weight"),
            optimise = FALSE, value = 1),
        run_at = g3_action_order$likelihood)

g3l_random_walk(
        nll_name,
        param_f,
        sigma_f = 1,
        log_f = TRUE,
        period = 'auto',
        nll_breakdown = FALSE,
        weight = g3_parameterized(paste0(nll_name, "_weight"),
            optimise = FALSE, value = 1),
        run_at = g3_action_order$likelihood)
```

Arguments

param_f	A formula representing the value to apply dnorm to. Invariably a g3_param for g3l_random_dnorm, a g3_param_table with *cur_year* for g3l_random_walk.
mean_f	A formula representing *mean* in dnorm.
sigma_f	A formula representing *sigma* in dnorm.
log_f	A formula representing *log* in dnorm.
period	When dnorm should be recalculated. Once per `year` every `step`, or `single` for once. The default, `auto`, will assume the input is generated by `g3_parameterized` and will derive the most appropriate option.
nll_name	Character string, used to define the variable name for *dnorm* output.
nll_breakdown	Should the nll report be broken down by time? `TRUE` / `FALSE`
weight	Weighting applied to this likelihood component.
run_at	Integer order that actions will be run within model, see `g3_action_order`.

Details

The model report will contain `nll_random_dnorm_dnorm_lin__dnorm`, the results of applying dnorm. If *nll_breakdown* is `TRUE`, this will be an array with one entry per timestep.

Value

g3l_random_dnorm: An action (i.e. list of formula objects) that will...

1. On the final model step, calculate `dnorm(param_f, mean_f, sigma_f)` & add to nll

g3l_random_walk: An action (i.e. list of formula objects) that will...

1. Calculate `dnorm(param_f, previous param_f, sigma_f)` (at final year if period = year)
2. Add to nll.

Examples

```
likelihood_actions <- list(
    # Calculate dnorm() for the dnorm_log parameter
    g3l_random_dnorm('dnorm_log',
        g3_parameterized('dnorm_log', value = 0, random = TRUE),
        mean_f = 0),

    # Treat the walk_year.xxxx parameters as a random walk
    g3l_random_walk('walk_year',
        g3_parameterized('walk_year', by_year = TRUE, value = 0, random = TRUE))
)
```

`likelihood_sparsesample`

Gadget3 likelihood actions for sparse data

Description

Compare model predictions against a set of sparse data points

Usage

```
g3l_sparsesample_linreg(
        fit = c('log', 'linear'),
        slope = 1,
        intercept = NULL )

g3l_sparsesample_sumsquares(
        weighting = "model_stddev" )

g3l_sparsesample(
        nll_name,
        obs_df,
        stocks,
        measurement_f = quote( wgt ),
        function_f = g3l_sparsesample_linreg(),
        predstocks = list(),
```

```
    area_group = NULL,
    weight = g3_parameterized(paste(
        if (length(predstocks) > 0) "csparse" else "asparse",
        function_f_name,
        nll_name,
        "weight",
        sep = "_"), optimise = FALSE, value = 1),
    run_at = g3_action_order$likelihood )
```

Arguments

slope, intercept	formula substituted into surveyindices calcuations to fix slope/intercept of linear regression, or NULL if not fixed. See below.
fit	Is the fit 'log' or 'linear'? See below.
weighting	Weighting applied to sum-of-squares. One of "model_stddev", "obs_stddev" or a formula.
nll_name	Character string, used to define the variable name for *obsstock* and *modelstock*. By default set to `(asparse|csparse)_(name_of_function_f)_(nll_name)_weight`.
obs_df	Data.frame of observation data. See details.
stocks	A list of `g3_stock` objects to collect sparsesample data for, depending if *stocks* were provided.
measurement_f	formula to derive the model's equivalent predicted value for a data point. You can use `wgt` to refer to weight of matching individuals, `length` to refer to length of matching individuals.
function_f	A formula to compare *obs_df* to predicted values generated via *transform_f* and generate nll, defined by one of the *g3l_sparsesample_** functions.
predstocks	A list of `g3_stock` predator or fleet objects. If present, we will compare against the model predicted catch. Without (the default), we compare against overall abundance.
area_group	List mapping area names used in *obs_df* to integer model areas, most likely generated by `g3_areas`.
weight	Weighting applied to this likelihood component. Default is a `g3_param` that defaults to 1, allowing weights to be altered without recompiling.
run_at	Integer order that actions will be run within model, see `g3_action_order`.

Details

The actions will define the following variables in your model, which could be reported with `g3a_report_history`:

nll_sp(abund|catch)_name__obs_mean Observation mean, the *mean* column from *obs_df*

nll_sp(abund|catch)_name__obs_stddev Observation standard deviation, the *stddev* column from *obs_df*

nll_sp(abund|catch)_name__obs_n Observation number, the *number* column from *obs_df*

nll_sp(abund|catch)_name__model_sum The corresponding model prediction vector, total data-points. `__model_sum / __model_n` for the mean

nll_sp(abund|catch)_name__model_sqsum The corresponding model prediction vector, sqared-sum datapoints.

nll_sp(abund|catch)_name__model_n The number of data points at each point in the model prediction vector, if *predstocks* set this is the number of individuals caught matching the datapoint (length/age/...), otherwise abundance of individuals matching the datapoint.

obs_df format: data.frame of observation data. Unlike `g3l_abundancedistribution`, gaps and sparse data is accepted, and gaps will not be filled with zero.

For each row in the table, all matching predictions are aggregated. Aggregation columns include:

year Required. The year the sample is from

step Optional. The timestep/season the sample is from

area Optional. Only aggregate predicted values from given area

age Optional. Only aggregate predicted values with given age

length Optional. Only aggregate predicted values with given length (matches nearest length-group)

So, a row with "year=1998,age=4" will be compared against age 4 individuals of all lengths in 1998, step 1 & 2. A row with "year=2004,step=1,age=2,length=19" will be compared against individuals of age 4, length 10..20, in winter 2004.

The observation data is provided in the following columns:

mean Required. Mean value at this data point

number Optional. Number of data points, defaults to 1

stddev Optional. Observed standard deviation (only required if `weighting = "obs_stddev"`)

Value

g3l_sparsesample_linreg: Returns a formula for use as *function_f*:

If `fit = "log"`:

$$\sum_{i}^{rows} (\alpha + \beta \log N_i - \log \frac{\nu_i}{P_i})^2$$

If `fit = "linear"`:

$$\sum_{i}^{rows} (\alpha + \beta N_i - \frac{\nu_i}{P_i})^2$$

N_i "mean" column from *obs_df*

ν_i Total predicted values for all data points, i.e. *nll_spabund_name__model_sum*

P_i Number of data points, i.e. *nll_spabund_name__model_n*

α *intercept* parameter, defaults to `1`, i.e. fixed slope

β *slope* parameter, defaults to `NULL`, i.e. linear regression performed to find optimal value

If either *alpha* or *beta* is not provided, then linear regression is performed on N vs ν for each value in table, and the optimal value used for each.

g3l_sparsesample_sumsquares: Returns a formula for use as *function_f*:

$$\sum_{i}^{rows} w(\frac{\nu_i}{P_i} - N_i)^2$$

N_i "mean" column from *obs_df*

ν_i Total predicted values, i.e. *nll_spabund_name__model_sum*

P_i Number of data points, i.e. *nll_spabund_name__model_n*

w *weighting* parameter, either:

1. $1/\sigma^2$, using stddev of model predicted values if `weighting = "model_stddev"`
2. $1/\sigma^2$, using stddev column from *obs_df* if `weighting = "obs_stddev"`
3. A custom forumla provided for *weighting*

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-like.html, g3l_catchdistribution g3_stock

Examples

```
st <- g3_stock("fish", c(10, 20, 30)) %>% g3s_age(3,5)

# Generate some random sparsesample samples
obs_df <- data.frame(
    # NB: No 1993, we don't have any samples for that year
    year = rep(c(1990, 1991, 1992, 1994), each = 2),
    step = 1:2 )
obs_df$age = floor(runif(nrow(obs_df), min = 3, max = 5.1))
obs_df$length = floor(runif(nrow(obs_df), min = 10, max = 50))
obs_df$mean = runif(nrow(obs_df), min = 10, max = 1000)

actions <- list(
    g3a_time(1990, 1994, c(6,6)),
    # Use otherfood to populate abundance / mean weight
    g3a_otherfood(st,
        quote( age * 100 + stock__minlen ),
        quote( cur_year * 1e5 + cur_step * 1e4 + 0 * stock__minlen ) ),
    g3l_sparsesample(
        "bt",
        obs_df,
        list(st),
        measurement_f = g3_formula(
          # Derive blubber thickness from length/weight
         ((wgt/(wmax.a * length^wmax.b) - 0.5) * 100 - 4.44) / (5693 * (length/wgt)^0.5),
          wmax.a = g3_parameterized("wmax.a", by_stock = TRUE),
          wmax.b = g3_parameterized("wmax.b", by_stock = TRUE),
          end = NULL ),
        function_f = g3l_sparsesample_linreg(fit = "linear") ),
    NULL )

model_fn <- g3_to_r(c(actions, list(
    g3a_report_detail(actions),  # TODO: Not reporting anything useful
    NULL )))
r <- attributes(model_fn())
colSums(r$dstart_fish__num)  # TODO: Report something related
```

`likelihood_tagging_ckmr` *Gadget3 CKMR likelihood*

Description

Experimental CKMR tagging likelihood

Usage

```
g3l_tagging_ckmr(
        nll_name,
        obs_data,
        fleets,
        parent_stocks,
        offspring_stocks,
        weight = g3_parameterized(paste0(nll_name, "_weight"),
            optimise = FALSE, value = 1),
        run_at = g3_action_order$likelihood)
```

Arguments

`nll_name`	Character string, used to define the variable name for *obsstock* and *modelstock*.
`obs_data`	Data.frame of observed mother-offspring pairs with columns year / parent_age / offspring_age / mo_pairs
`fleets`	A list of `g3_stock` objects to collect catch data for.
`parent_stocks`	A list of `g3_stock` objects that are parents in a g3a_spawn action
`offspring_stocks`	A list of `g3_stock` objects that are `output_stocks` in a g3a_spawn action
`weight`	Weighting applied to this likelihood component. Default is a `g3_param` that defaults to 1, allowing weights to be altered without recompiling.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.

Details

Implementation of CKMR based on *Bravington, M.V., Skaug, H.J., & Anderson, E.C. (2016). Close-Kin Mark-Recapture. Statistical Science, 31, 259-274.*

Only one kinship probability is implemented, mother-offspring with lethal sampling, i.e. (3.2) in the paper. This is then used as a pseudo-likelihood as per (4.1).

obs_data: The *obs_data* data.frame provides observed pairs. Unlike other likelihood mehthods, it has a fixed structure:

year Year of observation for the data point.

parent_age Age of the parent in an observed parent-offspring pair.

offspring_age Age of the offspring in an observed parent-offspring pair.

mo_pairs Number of pairs observed with these ages.

Value

g3l_tagging_ckmr: An action (i.e. list of formula objects) that will...

1. For all *parent_stocks* and *offspring_stocks*, collect spawing rate into *modelhist__*spawning and *modelhist__*spawned, total number of parents and total number of spawned offspring respectively
2. For all *fleets*, collect catch data into *modelhist__*catch
3. For any observed pairs that year, include the probability of that event happening into `nll`

See Also

Bravington, M.V., Skaug, H.J., & Anderson, E.C. (2016). Close-Kin Mark-Recapture. Statistical Science, 31, 259-274. `g3_stock`

`likelihood_understocking`

Gadget3 likelihood understocking action

Description

Add rates of understocking in a g3 model to nll

Usage

```
g3l_understocking(
        prey_stocks,
        power_f = ~2,
        nll_breakdown = FALSE,
        weight = 1e+08,
        run_at = g3_action_order$likelihood)
```

Arguments

`prey_stocks`	A list of `g3_stock` objects to collect catch data for
`power_f`	A formula representing power coefficient p to use.
`nll_breakdown`	Should the nll report be broken down by time? `TRUE` / `FALSE`
`weight`	Weighting applied to this likelihood component.
`run_at`	Integer order that actions will be run within model, see `g3_action_order`.

Details

The model report will contain nll_understocking__wgt, the results of the formula below. If *nll_breakdown* is `TRUE`, this will be an array with one entry per timestep.

Value

g3l_distribution_understocking: An action (i.e. list of formula objects) that will...

1. Sum the total biomass adjustment due to overstocking for each prey according to the formula

$$\ell = \sum_{time} \sum_{areas} \Big(\sum_{prey\_stocks} U_{trs} \Big)^{p}$$

Where p is the power coefficient from *power_f*, U_{trs} is the total biomass adjustment to predator consumtion due to overconsumtion.

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock('ling_mat', seq(20, 156, 4)) %>% g3s_age(5, 15)
lln <- g3_fleet('lln')

likelihood_actions <- list(
  g3l_understocking(list(ling_imm, ling_mat)))
```

`params` *Gadget3 parameter helpers*

Description

Shortcuts to parameterise a model with g3_param

Usage

```
g3_parameterized(
        name,
        by_stock = FALSE,
        by_predator = FALSE,
        by_year = FALSE,
        by_step = FALSE,
        by_age = FALSE,
        by_area = FALSE,
        prepend_extra = list(),
        exponentiate = FALSE,
        avoid_zero = FALSE,
        scale = 1,
        offset = 0,
        ifmissing = NULL,
        ...)
```

Arguments

`name`	Suffix for parameter name.
`by_stock`	Should there be individual parameters per-stock? **FALSE** No **TRUE** Produce a `"stock_name.name"` parameter **character vector** Select the stock name_part(s) to use, e.g. to produce `"stock_species.name"` parameter with "species" **List of `g3_stock` objects** Produce a parameter that applies to all given stocks
`by_predator`	Should there be individual parameters per-predator (read: per-fleet) stock? **FALSE** No **TRUE** Produce a `"fleet_stock_name.name"` parameter **character vector** Select the stock name_part(s) to use, e.g. to produce `"fleet_country.name"` parameter with "country" **List of `g3_stock` objects** Produce a parameter that applies to all given stocks
`by_year`	Should there be individual parameters per model year? **FALSE** No **TRUE** Produce a `"name.1998"` parameter for each year the model runs **1998:2099** Override the year range, so when projecting there will be sufficient parameters available.
`by_step`	Should there be individual parameters per step within years? **FALSE** No **TRUE** Produce a `"name.1"` seasonal parameter for each step, or `"name.1998.1"` for every timestep in the model if combined with *by_year*.
`by_age`	Should there be individual parameters per stock age? **FALSE** No **TRUE** Produce a `"name.4"` parameter for each age of the stock(s) in *by_stock*
`by_area`	Should there be individual parameters per stock area? **FALSE** No **TRUE** Produce a `"name.area"` parameter for each area of the stock(s) in *by_stock*
`prepend_extra`	List of extra things to prepend to the parameter name. Can be a string, or a stock object. See `stock_prepend`, which *prepend_extra* is passed to
`ifmissing`	Value to use for when there is no valid parameter (read: year when by_year = TRUE) Either a numeric constant or character. If character, add another parameter for ifmissing, using the same *by_stock* value.
`exponentiate`	Use `exp(value)` instead of the raw parameter value. Will add "_exp" to the parameter name.
`avoid_zero`	If TRUE, wrap parameter with `avoid_zero`
`scale`	Use `scale * value` instead of the raw parameter value. Either a numeric constant or character. If character, add another parameter for scale, using the same *by_stock* value.
`offset`	Use `value + offset` instead of the raw parameter value Either a numeric constant or character. If character, add another parameter for offset, using the same *by_stock* value.
`...`	Additional parameters passed through to `g3_param`, e.g. *optimise*, *random*, ...

Details

The function provides shortcuts to common formulas used when parameterising a model.

Value

A formula object defining the given parameters

See Also

g3_param, g3_param_table, stock_prepend

Examples

```
stock_imm <- g3_stock(c(species = 'stock', 'imm'), seq(10, 35, 5)) %>% g3s_age(1, 4)
stock_mat <- g3_stock(c(species = 'stock', 'mat'), seq(10, 35, 5)) %>% g3s_age(3, 6)

# Helper function that shows the parameter template for the given parameter
param_template_for <- function (g3_param) {
  model_code <- g3_to_tmb(list(
      # g3a_naturalmortality() isn't important, it is a place to add our parameter
      g3a_naturalmortality(stock_imm, g3_param),
      # We also need stock_mat in the model at least once
      g3a_naturalmortality(stock_mat, 0),

      # Set a year range to use for parameters where relevant
      g3a_time(1990, 1994) ))

  # Extract template, throw away default parameters from g3a_time()
  params <- attr(model_code, "parameter_template")
  params <- params[!(rownames(params) %in% c('retro_years', 'project_years')),]
  return(params)
}

# Not 'by' anything, so we add a single parameter value
param_template_for( g3_parameterized('K') )

# Can set defaults for the parameter template when defining a parameter
param_template_for( g3_parameterized('K', value = 5, lower = 2,
    upper = 8, optimise = FALSE) )

# by_stock, so the parameters will have the stock name prepended
param_template_for( g3_parameterized('K', by_stock = TRUE) )

# Similarly, we can prepend year & age
param_template_for( g3_parameterized('K', by_stock = TRUE, by_year = TRUE, by_age = TRUE) )

# You can specify the name part to use,
# e.g. if a parameter should be shared between mature and immature:
param_template_for( g3_parameterized('K', by_stock = 'species', by_year = TRUE) )

# Can give a list of stocks, in which case it works out name parts for you
param_template_for( g3_parameterized('K', by_stock = list(stock_imm, stock_mat)) )
```

```
param_template_for( g3_parameterized('K', by_stock = list(stock_imm, stock_mat), by_age = TRUE) )

# If there are no shared name parts, then all names will be added
param_template_for( g3_parameterized(
    'btrigger',
    by_stock = list(g3_fleet("surv"), g3_fleet("comm"))) )

# You can set fixed scale/offset for the parameter
g3_parameterized('K', scale = 5, offset = 9)

# ...or give names and they will also be parameters, sharing the by_stock setting
param_template_for( g3_parameterized('K', by_stock = TRUE, scale = "sc", offset = "offs") )
```

`param_project` *Gadget3 projected parameters*

Description

Add time-based random deviates / projections

Usage

```
g3_param_project_dlnorm(
        lmean_f = g3_parameterized("proj.dlnorm.mean",
            value = 1e-5, optimise = FALSE, type = "LOG",
            prepend_extra = quote(param_name) ),
        lstddev_f = g3_parameterized("proj.dlnorm.stddev",
            value = 0.2, optimise = FALSE, type = "LOG",
            prepend_extra = quote(param_name) ))

g3_param_project_dnorm(
        mean_f = g3_parameterized("proj.dnorm.mean",
            value = 0, optimise = FALSE,
            prepend_extra = quote(param_name) ),
        stddev_f = g3_parameterized("proj.dnorm.stddev",
            value = 1, optimise = FALSE,
            prepend_extra = quote(param_name) ))

g3_param_project_rwalk(
        mean_f = g3_parameterized("proj.rwalk.mean",
            value = 0, optimise = FALSE,
            prepend_extra = quote(param_name) ),
        stddev_f = g3_parameterized("proj.rwalk.stddev",
            value = 1, optimise = FALSE,
            prepend_extra = quote(param_name) ))

g3_param_project_ar1(
```

```
        phi_f = g3_parameterized(
            "proj.ar1.phi",
            value = 0.8, lower = 0, upper = 1, optimise = FALSE,
            prepend_extra = quote(param_name) ),
        stddev_f = g3_parameterized(
            "proj.ar1.stddev",
            value = 1, optimise = FALSE,
            prepend_extra = quote(param_name) ),
        level_f = g3_parameterized(
            "proj.ar1.level",
            value = 0,
            prepend_extra = quote(param_name) ),
        lastx_f = 0L)

g3_param_project_logar1(
        phi_f = g3_parameterized(
            "proj.logar1.phi",
            value = 0.8, lower = 0, upper = 1, optimise = FALSE,
            prepend_extra = quote(param_name) ),
        lstddev_f = g3_parameterized(
            "proj.logar1.stddev",
            value = 0.2, optimise = FALSE, type = "LOG",
            prepend_extra = quote(param_name) ),
        loglevel_f = g3_parameterized(
            "proj.logar1.level",
            value = 1, type = "LOG",
            prepend_extra = quote(param_name) ),
        lastx_f = 0L)

g3_param_project(
        param_name,
        project_fs = g3_param_project_rwalk(),
        by_step = TRUE,
        by_stock = FALSE,
        weight = g3_parameterized(
            paste("proj", project_fs$name, param_name, "weight", sep = "_"),
            optimise = FALSE, value = 1),
        scale = 1,
        offset = 0,
        random = TRUE )
```

Arguments

`mean_f`, `stddev_f`, `phi_f`, `lmean_f`, `lstddev_f`
: mean / stddev in normal / logspace used for both the likelihood of deviates & to project future values. Defaults to parameters with names `(by_stock).(param_name).proj.(mean|stdd`

`level_f`, `loglevel_f`
: (logspace) level (or offset) applied on top of ar1/logar1 regression. Defaults to parameter with name `(by_stock).(param_name).proj.(level|loglevel)`,

`lastx_f`	If > `0`, the setting of *level_f* / *loglevel_f* will be ignored, and the mean of the last (x) non-projection values are used as (log)level. Defaults to 0L, i.e. disabled.
`param_name`	Character string used to name the parameters.
`project_fs`	Results of either `g3_param_project_dnorm`, `g3_param_project_rwalk`.
`by_step`	Boolean, generate per-step or per-year values.
`by_stock`	Prepend stock name to the projection variable, i.e. *param_name*. Unlike `g3_parameterized`, can only be `FALSE` or `g3_stock` objects, `TRUE` or `"species"` isn't supported.
`weight`	A weighting to give to the likelihood when generating total nll.
`scale, offset`	Number, formula or string. Scale / offset to add to parameter values. If string, then the scale/offset will also be a parameter, equivalent to setting `scale = g3_parameterized(c(param_name, "proj", "scale"))`.
`random`	Boolean, tell TMB to treat the deviates as random variables by default. Can be changed in the parameter template.

Details

The actions will define the following variables in your model, which could be reported with `g3a_report_history`:

proj_(dnorm|rwalk)_(param_name)__var Vector of all values, both parameters & projected, by time

proj_(dnorm|rwalk)_(param_name)__lvar Vector of all values, both parameters & projected, by time (logarithmic scale)

proj_(dnorm|rwalk)_(param_name)__nll Likelihood of each value

Value

g3_param_project_dlnorm: Returns a "nll" & "project" formula objects for use as *project_fs*.
The functions compare / generate normally-distributed deviates around a mean, i.e:

$$V_t = \epsilon_{M-\frac{e^{2*\Sigma}}{2},\Sigma}$$

$$v = exp(V)$$

M *lmean_f* / `(by_stock).(param_name).proj.lmean` parameter

Σ *lstddev_f* / `(by_stock).(param_name).proj.lstddev` parameter

$\epsilon_{\mu,\sigma}$ Normally distributed noise generated using `rnorm`

v Output time series

nll Compare values against `dnorm(x, mean_f, stddev_f)`

proj Generate new values with `rnorm(mean_f, stddev_f)`

g3_param_project_dnorm: Returns a "nll" & "project" formula objects for use as *project_fs*.
The functions compare / generate log-normal deviates around a mean, i.e:

$$v_t = \epsilon_{\mu,\sigma}$$

μ *mean_f* / `(by_stock).(param_name).proj.mean` parameter

σ *stddev_f* / `(by_stock).(param_name).proj.stddev` parameter
$\epsilon_{\mu,\sigma}$ Normally distributed noise generated using `rnorm`
v Output time series

nll Compare values against `dnorm(x, mean_f, stddev_f)`
proj Generate new values with `rnorm(mean_f, stddev_f)`

g3_param_project_rwalk: Returns a "nll" & "project" formula objects for use as *project_fs*.
The functions compare / generate to a random walk, i.e:

$$v_t = v_{t-1} + \epsilon_{\mu,\sigma}$$

μ *mean_f* / `(by_stock).(param_name).proj.mean` parameter
σ *stddev_f* / `(by_stock).(param_name).proj.stddev` parameter
$\epsilon_{\mu,\sigma}$ Normally distributed noise generated using `rnorm`
v Output time series

nll Compare difference between values `dnorm(x, mean_f, stddev_f)`
proj Generate new values with a delta of `rnorm(mean_f, stddev_f)`

g3_param_project_ar1: Returns a "nll" & "project" formula objects for use as *project_fs*.
The functions compare / generate a AR1 process projecting from any existing values, i.e:

$$v_t = \phi v_{t-1} + (1-\phi)\theta + \epsilon_{0,\sigma}$$

ϕ *phi_f* / `(by_stock).(param_name).proj.phi` parameter
θ *level_f* / `(by_stock).(param_name).proj.level` parameter
σ *stddev_f* / `(by_stock).(param_name).proj.stddev` parameter, if 0 1e-7 is used, so we don't return Inf
$\epsilon_{\mu,\sigma}$ Normally distributed noise generated using `rnorm`
v Output time series

g3_param_project_logar1: Returns a "nll" & "project" formula objects for use as *project_fs*.
The functions compare / generate a log-AR1 process projecting from any existing values, i.e:

$$V_t = \Phi V_{t-1} + (1-\Phi)\Theta + \epsilon_{0-\frac{e^{2*\Sigma}}{2},\Sigma}$$

$$v = exp(V)$$

Φ *phi_f* / `(by_stock).(param_name).proj.phi` parameter
Θ *loglevel_f* / `(by_stock).(param_name).proj.loglevel` parameter
Σ *lstddev_f* / `(by_stock).(param_name).proj.lstddev` parameter, if 0 1e-7 is used, so we don't return Inf
$\epsilon_{\mu,\sigma}$ Normally distributed noise generated using `rnorm`
v Output time series

g3_param_project: Returns a formula to choose the current value from the `__var` / `__lvar` vector.
An extra G3 action will:

1. Populate the array with random deviates from parameters (see examples)
2. Project for any projection years (see g3a_time)
3. Add likelihood comparing random deviates to expected values

g3l_sparsesample_sumsquares: Returns a formula for use as *function_f*:

$$\sum_{i}^{rows} w(\frac{\nu_i}{P_i} - N_i)^2$$

N_i "mean" column from *obs_df*

ν_i Total predicted values, i.e. *nll_spabund_name__model_sum*

P_i Number of data points, i.e. *nll_spabund_name__model_n*

w *weighting* parameter, either:

1. $1/\sigma^2$, using stddev of model predicted values if `weighting = "model_stddev"`
2. $1/\sigma^2$, using stddev column from *obs_df* if `weighting = "obs_stddev"`
3. A custom forumla provided for *weighting*

See Also

g3a_time g3_parameterized

Examples

```
st <- list(
    imm = g3_stock(c("fish", maturity = "imm"), c(10, 20, 30)),
    mat = g3_stock(c("fish", maturity = "mat"), c(10, 20, 30)) )
st2 <- g3_stock("other", c(10, 20, 30))

# Set up a projected parameter to share over both stocks
st_Mdn <- g3_param_project(
   "Mdn",
   g3_param_project_dnorm(),
   # Append common part of stock names to parameter name
   by_stock = st )

actions <- list(
    g3a_time(1990, 1994, c(6,6)),
    gadget3:::g3a_initialconditions_manual(st$imm,
        quote( 100 + stock__minlen ),
        quote( 1e4 + 0 * stock__minlen ) ),
    gadget3:::g3a_initialconditions_manual(st$mat,
        quote( 100 + stock__minlen ),
        quote( 1e4 + 0 * stock__minlen ) ),
    gadget3:::g3a_initialconditions_manual(st2,
        quote( 100 + stock__minlen ),
        quote( 1e4 + 0 * stock__minlen ) ),

    # Natural mortality with per-step deviates
    g3a_naturalmortality(st$imm, g3a_naturalmortality_exp(st_Mdn)),
    g3a_naturalmortality(st$mat, g3a_naturalmortality_exp(st_Mdn)),
```

```
    # Natural mortality with per-year random walk
    g3a_naturalmortality(st2, g3a_naturalmortality_exp(
        g3_param_project(
          "Mrw",
          g3_param_project_rwalk(),
          # The same value will be used for each step
          by_step = FALSE,
          # by_stock means the stock name will be included in parameter names
          by_stock = st2 ))),
    NULL )

model_fn <- g3_to_r(c(actions, list(
    g3a_report_history(actions, 'proj_.*', out_prefix = NULL),
    NULL )))

# Mdn has a parameter for each year/step, as well as mean/sd (added above) & likelihood weighting
grep("^fish.Mdn", names(attr(model_fn, 'parameter_template')), value = TRUE)

# Mrw has parameters for each year
grep("^other.Mrw", names(attr(model_fn, 'parameter_template')), value = TRUE)

attr(model_fn, 'parameter_template') |>
    g3_init_val("stst.Mdn.#.#", 0.5, lower = 0.1, upper = 0.9, random = TRUE) |>
    g3_init_val("stst.Mdn.proj.dnorm.lmean", 0.1) |>
    g3_init_val("stst.Mdn.proj.dnorm.lstddev", 0.001) |>

    g3_init_val("other.Mrw.proj.rwalk.mean", 0) |>
    g3_init_val("other.Mrw.proj.rwalk.stddev", 0.001) |>
    g3_init_val("other.Mrw.#", 0.5, lower = 0.1, upper = 0.9, random = TRUE) |>

    # Project forwards 20 years
    g3_init_val("project_years", 20) |>

    # Don't include projections in nll calculations:
    # allows a stddev to be supplied for projections, but estimated freely
    g3_init_val("proj_rwalk_fish_Mrw_weight", 0) |>
    g3_init_val("proj_dnorm_fish_Mdn_weight", 0) |>

    identity() -> params
r <- attributes(model_fn(params))

# Values used for dnorm
plot(r$proj_dnorm_fish_Mdn__var)

# Values used for random walk
plot(r$proj_rwalk_other_Mrw__var)

### Plot values for an individual projection function

actions <- list( g3a_time(1990, 1991), g3_param_project("M", g3_param_project_dlnorm()) )
model_fn <- g3_to_r(c(actions, list(
    g3a_report_history(actions, 'proj_.*', out_prefix = NULL),
```

```
    NULL )))

attr(model_fn, 'parameter_template') |>
    g3_init_val("M.proj.dlnorm.lmean", log(20)) |>
    g3_init_val("M.proj.dlnorm.lstddev", log(1e-6)) |>
    g3_init_val("M.#.#", 20) |>

    g3_init_val("project_years", 100) |>

    identity() -> params

par(mfrow=c(3, 1))
plot(attr(model_fn(params |>
   g3_init_val("M.proj.dlnorm.lstddev", log(1.001)) ), "proj_dlnorm_M__lvar"), ylim = c(15, 25))
plot(attr(model_fn(params |>
   g3_init_val("M.proj.dlnorm.lstddev", log(1e-1)) ), "proj_dlnorm_M__lvar"), ylim = c(15, 25))
plot(attr(model_fn(params |>
   g3_init_val("M.proj.dlnorm.lstddev", log(1e-2)) ), "proj_dlnorm_M__lvar"), ylim = c(15, 25))
```

---

quota *Gadget3 projected quotas*

Description

Add projected fishing quotas / MSE

Usage

```
g3_quota_hockeystick(
        predstocks,  # Predator / fleet stocks forming a name for quota
        preystocks,  # Mature spawning-stocks
        preyprop_fs = 1,  # NB: Doesn't have to sum to 1
        trigger = g3_parameterized("hs.trigger", by_stock = predstocks),
        target = g3_parameterized("hs.target", by_stock = predstocks),
       stddev = g3_parameterized("hs.stddev", by_stock = predstocks, value = 0),
        unit = c("harvest-rate-year", "biomass-year", "individuals-year") )

g3_quota_hockeyfleet(
        predstocks,  # Predator / fleet stocks forming a name for quota
        preystocks,  # Mature spawning-stocks
        preyprop_fs = 1,  # NB: Doesn't have to sum to 1
        btrigger = g3_parameterized("hf.btrigger", by_stock = predstocks),
      harvest_rate = g3_parameterized("hf.harvest_rate", by_stock = predstocks),
       stddev = g3_parameterized("hf.stddev", by_stock = predstocks, value = 0) )

g3_quota_assess(
        predstocks,
        preystocks,
```

```
        assess_f,
        unit = c("biomass-year", "biomass", "harvest-rate", "harvest-rate-year",
                 "individuals", "individuals-year") )

g3_quota(
        function_f,
        quota_name = attr(function_f, 'quota_name'),
        init_val = NaN,
        year_length = 1L,
        start_step = 1L,
        run_revstep = -1,
        run_historical = FALSE,
        run_f = TRUE,
        run_at = g3_action_order$quota )
```

Arguments

predstocks	A list of g3_stock objects for all predators/fleets that will use the quota. In g3_quota_hockeyfleet, these will be used to name the quota/parameters. In g3_quota_assess, these will define helper variables for use in *function_f*.
preystocks	A list of g3_stock objects for all prey that the quota will apply to. In g3_quota_hockeyfleet, these will be used to define the spawning stock biomass. In g3_quota_assess, these will define helper variables for use in *function_f*.
preyprop_fs	A formula or list of formulas representing the proportion of that prey that makes up the Spawning Stock Biomass (SSB). The proportions do not need to sum to 1, for example you may use `preyprop_fs = 0.4` to assume that 40 Using a suitability function is also supported, e.g. g3_suitability_exponentiall50.
trigger, btrigger	Trigger biomass (or number of individuals, if *unit* is "individuals-year"), see formula
target, harvest_rate	The maximum quota value to be returned, assuming the stock is healthy, and above *trigger*, see formula
stddev	If > 0, then apply log-normal noise to the output quota.
assess_f	A formula that runs an assessment model & returns a quota. See vignette TODO:
unit	A single string representing the returned quota's unit, as used by g3a_predate_catchability_project. In g3_quota_hockeyfleet, the unit of both *harvest_rate* and *btrigger* In g3_quota_assess, the value returned by *assess_f* will be assumed to have this unit.
function_f	Output of one of the g3_quota_* functions, responsible for choosing the next quota
quota_name	A name used to refer to the quota internally, by default a combination of the quota function and the stocks used.
init_val	Pre-fill the quota with this value on the start of the model, will be used if no quota assessment has run for the current fishing year. This needs to be NaN for g3a_predate_catchability_project to detect interim periods.
year_length	The length of the *fishing year*, in years, see details.

`start_step` The initial step of the *fishing year*, in model steps, see details. This can be used to offset the fishing year from the calendar year for, if your fishing year should run autumn–autum. It can also offset from the start of the model, if your model starts at 1998 and your fishing year should run 2000–2005.

`run_historical` Boolean. Should the quota be calculated for historical periods? Default `FALSE`, assuming `g3a_predate_catchability_project` will be supplying landings data and any quota would be unused. Requied to be FALSE to detect interim periods.

`run_revstep` A negative integer, defining which step in the *fishing year* an assessment for next year is performed. If `NULL`, run every step.

`run_f` When the quota should be recalculated, in addition to any condition defined by *run_revstep*.

`run_at` Integer order that actions will be run within model, see `g3_action_order`.

Details

Variables defined in your model: Once added the following variables can be reported:

quota_*quota_name*__**var** A vector of quota values, one per *fishing year* in your model, see `quota_hockeyfleet_trawl__var` in example below

Fishing year: Instead of generating a quota per calendar year or step, as we do with other projections, quotas are per fishing year.

The schedule of the fishing calendar is defined with:

year_length The length of the fishing year, in years. If > 1, the same yearly quota will be used for all years

start_step Offset of the initial fishing year, in model steps. So you can both start your fishing year in autumn, and have a fishing year that is offset against the model start year

run_revstep The step in the fishing year that the quota for the next year should be calculated

In addition, `g3a_predate_catchability_project` will allow you to assign proportions of a quota to model steps.

Examples:

`year_length = 2, start_step = 3, run_revstep = -2` A 2 year fishing calendar, i.e. a quota will be calculated every other year and the same value used for the next 2 years. Assuming 4 model steps, the quota will be calculated in winter for the next fishing year in summer.

`year_length = 5, start_step = 4 * 2` A 5 year fishing calendar, the quota will be recalulated every 5 years in the autumn (the final step before the next fishing year starts). If our model starts at 1998 and have 4 steps, `4 * 2` means our first full fishing year is 2000–2005.

Value

g3_quota_hockeystick: A formula for use in `g3_quota`

$$SS = \sum_{p}^{preys} S_p N_p$$

... if unit = `"individuals-year"`

$$SS = \sum_{p}^{preys} S_p N_p W_p$$

... otherwise

$$Tgmin(\frac{SS}{Tr}, 1)$$

Tg Target consumption, provided by *target* argument, by default the `hs.target` parameter

Tr Trigger biomass / harvest-rate / individuals, provided by *trigger* argument, by default the `hs.trigger` parameter

SS Spawning Stock (SS) in biomass / individuals

S_p Suitable proportion of prey p, as decided by *preyprop_fs*

N_p Total abundance of prey p

N_p Mean weight of prey p

g3_quota_hockeyfleet: For backward-compatibility, essentially the same as `g3_quota_hockeystick(..., unit = "harvest-rate-year")`

g3_quota_assess: A formula for use in `g3_quota`

g3_quota: A formula for use in `g3a_predate_catchability_project`, returning the current value of the quota `quota_`*quota_name*`__var` time vector.

In addition, returns an ancillary step that populates the `quota_`*quota_name*`__var` time vector according to the fishing year.

See Also

`g3a_predate_catchability_project`

Examples

```
st <- g3_stock("st", c(10))
fleets <- list(
    # NB: We break down our fleet names into parts, g3_quota_hockeystick() will
    # use the common name part (read: trawl) when naming parameters.
    nor = g3_fleet(c(type = "trawl", country = "nor")),
    oth = g3_fleet(c(type = "trawl", country = "oth")) )

# Define quota for both fleets, with an assessment in spring, application in autumn
fl_quota <- g3_quota(
    g3_quota_hockeystick(fleets, list(st), preyprop_fs = 1, unit="harvest-rate"),
    start_step = 4L,
    run_revstep = -2L )

# Invent some historical landings tables for the sake of example
landings_trawl_nor <- expand.grid(year = 1990:1995, step = 2)
landings_trawl_nor$total_weight <- 1e6
landings_trawl_oth <- expand.grid(year = 1990:1995, step = 2)
landings_trawl_oth$total_weight <- 1e8
```

```
actions <- list(
    g3a_time(1990, 1995, c(3,3,3,3)),
    # Define st with steadily collapsing stock
    g3a_otherfood(st, num_f = g3_timeareadata('st_abund', data.frame(
        year = 1990:2050,
        abund = 1e6 - 1e4 * seq(0, 2050-1990)), "abund"), wgt_f = 10),
    # Fleet predation, both sharing the same quota
    g3a_predate(
        fleets$nor,
        list(st),
        suitabilities = 0.8,
        catchability_f = g3a_predate_catchability_project(
            # Use the (shared) quota when projecting, otherwise use historical landings
            quota_f = fl_quota,
            landings_f = g3_timeareadata("landings_trawl_nor", landings_trawl_nor) )),
    g3a_predate(
        fleets$oth,
        list(st),
        suitabilities = 0.8,
        catchability_f = g3a_predate_catchability_project(
            # Use the (shared) quota when projecting, otherwise use historical landings
            quota_f = fl_quota,
            landings_f = g3_timeareadata("landings_trawl_oth", landings_trawl_oth) )),
    NULL )
model_fn <- g3_to_r(c(actions,
    g3a_report_detail(actions),
   g3a_report_history(actions, "__num$|__wgt$", out_prefix="dend_"),  # NB: Late reporting
    g3a_report_history(actions, "quota_", out_prefix = NULL) ))

attr(model_fn, "parameter_template") |>
    # Project for 30 years
    g3_init_val("project_years", 30) |>
    # Quota is yearly, so specify cons.step parameters to divide up into steps
    # Fishing predominantly occurs in spring/summer, none in winter
    g3_init_val("trawl_*.cons.step.#", c(0.0, 0.5, 0.4, 0.1)) |>
    # 3/4 of the quota goes to nor, the rest to oth
    g3_init_val("trawl_nor.quota.prop", 0.75) |>
    g3_init_val("trawl_oth.quota.prop", 0.25) |>
    # Hockefleet: harvest rate & trigger biomass (shared across trawl_nor & trawl_oth)
    g3_init_val("trawl.hs.target", 0.2) |>
    g3_init_val("trawl.hs.trigger", 7.2e6) |>
    identity() -> params.in
r <- attributes(model_fn(params.in))

## Total biomass at assessment point
g3_array_agg(r$dend_st__num * r$dend_st__wgt, "year", step = 2)

## Quota values, inflection once total biomass falls below btrigger
par(mar = c(6, 5, 1, 0)) ; barplot(r$quota_hockeystick_trawl__var, las = 2) ; abline(v=27.7)

## Consumption by fleet, demonstrating
##   (a) fixed landings before projections (landings_trawl_nor)
```

```
##   (b) inflection of hitting btrigger
##   (c) Uneven spread of fishing effort throughout year (fl.quota.step.#)
barplot(g3_array_agg(r$detail_st_trawl_nor__cons, "time"), las = 2)

## Timing of calculations for fishing year
fl_quota <- g3_quota(
    # Our quota values are year/step at the assessment time step
    quote( cur_year * 10 + cur_step ),
   # "init_val" will be used in the interim period, replaced by g3a_predate_catchability_project()
    init_val = 99,
    year_length = 1L,
    start_step = 4L,
    run_revstep = -3L )
yr <- as.integer(format(Sys.Date(), "%Y"))
actions <- list(
    g3a_time(yr - 6, yr - 1, project_years = 10, step_lengths = rep(3L, 4)),
    fl_quota,
   # At each step in the model, print the current year/step, and the quota value that will get used
   # NB: before projection, g3a_predate_catchability_project() will use landings data not the quota
    g3_step(g3_formula(
       writeLines(paste(cur_year, cur_step, if (cur_year_projection) q else "landings")),
        q = fl_quota )),
    NULL)
model_fn <- g3_to_r(c(actions,
    g3a_report_history(actions, "quota_", out_prefix = NULL) ))
attr(model_fn(), "quota__var")
```

`run_desc` *Gadget3 actions into R code*

Description

Convert g3 actions into a character vector describing the model

Usage

```
g3_to_desc(actions, minor_steps = FALSE)
```

Arguments

`actions`	A list of actions (i.e. list of formula objects), as produced by *g3a_** functions.
`minor_steps`	Include minor steps (e.g. zeroing cumulative arrays)? `TRUE` / `FALSE`

Value

Character vector describing each step in the model. An action in a model may have generated multiple steps (e.g. select each prey stock, scale total amount, apply overstocking), and there will be a line in here for each.

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)

initialconditions_action <- g3a_initialconditions_normalparam(
    ling_imm,
    by_age = TRUE)

# Timekeeping action
time_action <- g3a_time(
    start_year = 2000,
    end_year = 2004,
    c(3, 3, 3, 3))

# Generate a list outlining the steps the model uses
as.list(g3_to_desc(list(initialconditions_action, time_action)))
```

---

run_r *Gadget3 actions into R code*

Description

Convert g3 actions into an R function that can then be executed

Usage

```
g3_to_r(
        actions,
        work_dir = getOption('gadget3.r.work_dir', default = tempdir()),
        trace = FALSE,
        strict = FALSE,
        cmp_options = list(optimize = 3) )

## S3 method for class 'g3_r'
print(x, ..., with_environment = FALSE, with_template = FALSE)
```

Arguments

actions	A list of actions (i.e. list of formula objects), as produced by *g3a_** functions.
work_dir	Where to write the temporary R script containing your function
cmp_options	options to pass through to `compiler::cmpfun()`. If NULL, then don't run the model through the byte-code compiler
trace	If TRUE, turn all comments into print statements.
strict	If TRUE, enable extra sanity checking in actions. Any invalid conditions (e.g. more/less fish after growth) will result in a warning.
x	The g3_to_r-generated function to print

`with_environment`
: If TRUE, list data stored in function environment when printing

`with_template`
: If TRUE, show parameter template when printing

`...`
: Other arguments

Value

A function that takes a *params* variable, which can be:

1. A list of parameters as defined by `attr(fn, 'parameter_template')`
2. A data.frame of parameters defined by `g3_to_tmb`'s parameter template
3. Not provided, in which case the parameter defaults are used

The function will have the following attributes:

actions The original *actions* list given to the function

parameter_template A list of all parameters expected by the model, to fill in

Use e.g. `attr(fn, 'parameter_template')` to retrieve them.

Invariant model data will be stored as a closure, i.e. in `environment(fn)`. This can be fetched with `environment(fn)$cdist_sumofsquares_ldist_gil_obs__num`.

The function will return *nll* produced by the model. You can also use `attributes(nll)` to get any report variables from the model.

Examples

```
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4)) %>% g3s_age(3, 10)

initialconditions_action <- g3a_initialconditions_normalparam(
    ling_imm,
    factor_f = g3a_renewal_initabund(by_stock_f = 'species'),
    by_stock = 'species',
    by_age = TRUE)

# Timekeeping action
time_action <- g3a_time(
    start_year = 2000,
    end_year = 2004,
    c(3, 3, 3, 3))

# Generate a model from the above 2 actions
# NB: Obviously in reality we'd need more actions
fn <- g3_to_r(list(initialconditions_action, time_action))

if (interactive()) {
  # Edit the resulting function
  fn <- edit(fn)
}

param <- attr(fn, 'parameter_template')
```

```
param$project_years <- 0
param$ling.init.F <- 0.4
param$ling.Linf <- 160
param$ling.K <- 90
param$ling.recl <- 12
param$recage <- g3_stock_def(ling_imm, 'minage')
param[grepl('^ling.init.sd.', names(param))] <- 50.527220
param[grepl('^ling_imm.init.\\d+', names(param))] <- 1
param$ling_imm.init.scalar <- 200
param$ling_imm.walpha <- 2.27567436711055e-06
param$ling_imm.wbeta <- 3.20200445996187
param$ling_imm.M <- 0.15

# Run the model with the provided parameters
nll <- fn(param)

# Get the report from the last model run
report <- attributes(nll)

# Fetch a value from the model data
environment(fn)$ling_imm__midlen
```

---

run_tmb *Gadget3 actions into TMB code*

Description

Turn g3 actions into CPP code that can be compiled using TMB

Usage

```
g3_to_tmb(actions, trace = FALSE, strict = FALSE)

g3_tmb_adfun(
        cpp_code,
        parameters = attr(cpp_code, 'parameter_template'),
        compile_flags = getOption('gadget3.tmb.compile_flags', default =
            if (.Platform$OS.type == "windows") c("-O1", "-march=native")
            else c("-O3", "-flto=auto", "-march=native") ),
        work_dir = getOption('gadget3.tmb.work_dir', default = tempdir()),
        output_script = FALSE,
        compile_args = list(
           framework = getOption("gadget3.tmb.framework", default = "TMBad") ),
        ...)

g3_tmb_fn(
        cpp_code,
        def.parameters = attr(cpp_code, 'parameter_template'),
```

```
        ... )

g3_tmb_par(parameters, include_random = TRUE)

g3_tmb_lower(parameters)

g3_tmb_upper(parameters)

g3_tmb_parscale(parameters)

g3_tmb_relist(parameters, par)
```

Arguments

actions	A list of actions (i.e. list of formula objects), as produced by *g3a_** functions.
trace	If TRUE, turn all comments into print statements.
strict	If TRUE, enable extra sanity checking in actions. Any invalid conditions (e.g. more/less fish after growth) will result in a warning.
cpp_code	cpp_code as produced by *g3_to_tmb*.
parameters, def.parameters	Parameter table as produced by `attr(g3_to_tmb(...), 'parameter_template')`, modified to provide initial conditions, etc.
par	Parameter vector, as produced by one of 1. `nlminb(...)$par` 2. `obj.fun$env$last.par` 3. `g3_tmb_par()` The first will not include random parameters by default, the others will.
include_random	Should random parameters assumed to be part of par? Should be TRUE if using `obj.fun$fn`, `obj.fun$report` directly, e.g. `obj.fun$fn(g3_tmb_par(param_tbl))`. In other cases, FALSE.
compile_flags	List of extra flags to compile with, use e.g. "-g" to enable debugging output. Can be set with an option, e.g. `options(gadget3.tmb.compile_flags = c('-O0', '-g'))`
compile_args	Any other arguments to pass to TMB::compile
work_dir	Directory to write and compile .cpp files in. Defaults to R's current temporary directory Set this to preserve compiled output and re-use between R sessions if possible. Can be set with an option, e.g. `options(gadget3.tmb.work_dir = fs::path_abs('tmb-workdir'))`
output_script	If TRUE, create a temporary R script that runs MakeADFun, and return the location. This can then be directly used with gdbsource or `callr::rscript`.
...	Any other options handed directly to MakeADFun or `g3_tmb_adfun` (for `g3_tmb_fn`).

Details

g3_tmb_adfun: g3_tmb_adfun will do both the compile and MakeADFun steps of making a model. If the code is identical to an already-loaded model then it won't be recompiled, so repeated calls to g3_tmb_adfun to change *parameters* are fast.

If MakeADFun is crashing your R session, then you can use *output_script* to run in a separate R session. Use this with gdbsource to debug your model.

g3_tmb_fn: Wraps g3_tmb_adfun to produce a function suitable for single model runs or projection, like g3_to_r. *optimise* & *random* are ignored, instead all values from the provided parameters will be used when calling the function.

Internally it uses `obj.fn$simulate`, so the R RNG is updated after the run is finished (i.e. successive runs will produce different answers).

Value

g3_to_tmb: A string of C++ code that can be used as an input to *g3_tmb_adfun*, with the following attributes:

actions The original *actions* list given to the function

model_data An environment containing data attached to the model

parameter_template A data.frame to be filled in and used as *parameters* in the other `g3_tmb_*` functions

Use e.g. `attr(cpp_code, 'parameter_template')` to retrieve them.

g3_tmb_adfun: An ADFun as produced by TMB's MakeADFun, or location of temporary script if *output_script* is TRUE

g3_tmb_fn: An R function with the signature `function (par = NULL)`, where: *par* can be `NULL` (use default parameter values), a parameter template `data.frame`, or a parameter list.

Returns a list with all report variables.

g3_tmb_par: Values extracted from *parameters* table converted into a vector of values for `obj$fn(par)` or `nlminb`

g3_tmb_lower: Lower bounds extracted from *parameters* table converted into a vector of values for `nlminb`. Random parameters are always excluded

g3_tmb_upper: Lower bounds extracted from *parameters* table converted into a vector of values for `nlminb`. Random parameters are always excluded

g3_tmb_parscale: Parscale extracted from *parameters* table, converted into a vector of values for `nlminb`. Random parameters are always excluded

g3_tmb_relist: The *parameters* table value column, but with optimised values replaced with contents of *par* vector. i.e. the inverse operation to g3_tmb_par. *par* can either include or discount random variables.

Examples

```
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4)) %>% g3s_age(3, 10)

initialconditions_action <- g3a_initialconditions_normalparam(
    ling_imm,
    factor_f = g3a_renewal_initabund(by_stock_f = 'species'),
    by_stock = 'species',
    by_age = TRUE)

abundance_action <- g3l_abundancedistribution(
    'abundance',
    data.frame(year = 2000:2004, number = 100),
    stocks = list(ling_imm),
    function_f = g3l_distribution_sumofsquares())

# Timekeeping action
time_action <- g3a_time(
    start_year = 2000,
    end_year = 2004,
    c(3, 3, 3, 3))

# Generate a model from the above 2 actions
# NB: Obviously in reality we'd need more actions
cpp <- g3_to_tmb(list(initialconditions_action, abundance_action, time_action))

if (interactive()) {
  # Edit the resulting code
  cpp <- edit(cpp)
}

# Set initial conditions for parameters
attr(cpp, 'parameter_template') |>
    g3_init_val("project_years", 0) |>
    g3_init_val("ling.init.F", 0.4) |>
    g3_init_val("ling.Linf", 160) |>
    g3_init_val("ling.K", 90) |>
    g3_init_val("ling.t0", 0) |>
    g3_init_val("ling.init.sd.#", 50.527220) |>
    g3_init_val("ling_imm.init.#", 1, lower = 0, upper = 1000) |>
    g3_init_val("ling_imm.init.scalar", 200) |>
    g3_init_val("ling_imm.walpha", 2.275e-06) |>
    g3_init_val("ling_imm.wbeta", 3.2020) |>
    g3_init_val("ling_*.M.#", 0.15) |>
    identity() -> tmb_param




if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # Compile to a TMB ADFun
  tmb <- g3_tmb_adfun(cpp, tmb_param)
}
```

```
# NB: TMB::gdbsource() requires both "R" and "gdb" to be available
# NB: gdbsource hangs on windows - https://github.com/kaskr/adcomp/issues/385
if (all(nzchar(Sys.which(c('gdb', 'R')))) && .Platform$OS.type !="windows") {

  cpp_broken <- g3_to_tmb(list(
    initialconditions_action,
    abundance_action,
    g3_formula(quote( stop("This model is broken") )),
    time_action))

  attr(cpp_broken, 'parameter_template') |>
    g3_init_val("ling_imm.init.#", 1, lower = 0, upper = 1000) |>
    identity() -> params_broken

  # Build the model in an isolated R session w/debugger
  writeLines(TMB::gdbsource(g3_tmb_adfun(
      cpp_broken,
      params_broken,
      compile_flags = "-g",
      output_script = TRUE)))

}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # Perform a single run, using values in table
  result <- tmb$fn(g3_tmb_par(tmb_param))
}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # perform optimisation using upper/lower/parscale from table
  fit <- optim(tmb$par, tmb$fn, tmb$gr,
      method = "L-BFGS-B",
      upper = g3_tmb_upper(tmb_param),
      lower = g3_tmb_lower(tmb_param),
      control = list(maxit=10, parscale=g3_tmb_parscale(tmb_param)))
}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # perform optimisation without bounds
  fit <- optim(tmb$par, tmb$fn, tmb$gr)
}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # Go back to a list of parameters, suitable for the R version
  # NB: This will not set the values for random parameters
  param_list <- g3_tmb_relist(tmb_param, fit$par)
}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # Update parameters with values from last run, *including* random parameters.
  param_list <- g3_tmb_relist(tmb_param, tmb$env$last.par)
}
```

```
if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # Rebuild a simple function version for projection
  proj.fn <- g3_tmb_fn(cpp)

  # Get the reported results from a single run
  result <- proj.fn(tmb_param)
}

if (!( nzchar(Sys.getenv('GITHUB_CI')) && .Platform$OS.type == "windows" )) {
  # We can do similar to g3_tmb_fn() by using g3_tmb_adfun() directly

  # Rebuild, only including "Fun" (i.e. without auto-differentiation)
  # Result will only work for tmb$report
  # NB: Only parameters with optimise=TRUE can be varied
  obj.fn <- g3_tmb_adfun(cpp, tmb_param, type = "Fun")
  result <- obj.fn$report(g3_tmb_par(tmb_param, include_random = TRUE))

  # We can also use obj.fn$simulate
  result <- obj.fn$simulate(g3_tmb_par(tmb_param, include_random = TRUE))

  # NB: Before running simulations, you should use set.seed() to ensure random output
}
```

`step` *G3 stock_* transformation functions*

Description

Additional meta-functions to help manage writing stock-handling actions.

Usage

```
g3_step(step_f, recursing = FALSE, orig_env = environment(step_f))
```

Arguments

`step_f`	Input formula containing references to functions below
`recursing`	Only use default value
`orig_env`	Only use default value

Details

All action producing functions will run their output through `g3_step`. This means that the functions described here will be available in any gadget3 code.

They handle translation of stock instance naming, so code can refer to e.g. `stock__num` without having to translate naming to the final stock name, and iterating over stock dimensions.

Value

g3_step: A formula object with references to above functions replaced.

debug_label

Add a comment to the code to act as a label for that step, when producing an outline of the model. There shouldn't be more than one `debug_label` call in a step.

Models compiled with `trace = TRUE` will print the resultant string to stdout.

Arguments: Any number of character strings, or g3_stock variables. The latter will be replaced with the final name.

debug_trace

Identical to debug_label, but not considered a "label", just a code comment, so any number of calls can be added.

stock_assert

```
stock_assert(expression, message, message/stock-var, ...)
```

Assert that *expression* is true, if not abort with a message.

stock_reshape

```
stock_reshape(dest_stock, expression)
```

Output *expression* with it's length structure reshaped to match *dest_stock*. The source stock is considered to be the first one found in *expression*

How this is achieved depends on the difference. If the source and destination match then this is a no-op. Otherwise a transformation matrix is generated and included into the model.

stock_ss

```
stock_ss(stock_var, [ dimname = override, dimname = override, ... ][, vec = (dimname|full|single)
])
```

Subsets *stock_var* for the current iteration of stock_iterate().

The *vec* parameter decides the start value for all dimensions If `full`, no other dimensions are set. If set to a dimname, all dimensions after that dimension are set (i.e. a dimname-vector will be returned) If `single`, all dimensions are set (i.e. a single value wil be returned). The default is `length` if a length dimension is present (i.e. a length vector will be returned), otherwise `single`.

If *dimname*s are supplied, then the code supplied will override the above. This code can include `default`, which will be substituted for the default subset, or `missing` to represent an empty position in the subset. If a *dimname* is not present in *stock_var*, it will be ignored.

stock_ssinv

```
stock_ssinv(stock_var, [ dimname, dimname, ... ])
```

like stock_ss(), but subset only the mentioned *dimname*s.

stock_switch

`stock_switch(stock, stock_name1 = expr, stock_name2 = expr, ... [ default ])`

Switch based on name of *stock*, returning the relevant *expr* or *default*. If no default supplied, then an unknown stock is an error.

expr is implicitly wrapped with `stock_with(stock, ...)`, so any references to the stock variable will work. If only default is provided, then this is identical to calling `stock_with`.

stock_with

`stock_with(stock, expr)`

Replaced with *expr* but with all stock variables of *stock* renamed with their final name. This is generally needed when not iterating over a stock, but e.g. zeroing or summing the whole thing.

stock_iterate

`stock_iterate(stock, expr)`

Wrap *expr* with the code to iterate over vector dimensions in *stock*, accessed using `stock_ss(stock)`.

Which dimensions are iterated over is decided based on the call to `stock_ss(stock)`. By default, stock_ss leaves length blank so will iterate over a length vector for each dimension.

You can iterate over each value individually with the following: `stock_iterate(stock, stock_ss(stock, length = default) )`

Current values for each dimension will be available as variables, e.g. `area`, `age`, and can be used in formulae.

stock_intersect

`stock_intersect(stock, expr)`

Wrap *expr* with the code to intersect all dimensions with the dimensions of an outer stock_iterate().

stock_interact

`stock_interact(stock, expr, prefix = prefix)`

Wrap *expr* with the code to interact with the dimensions of an outer stock_iterate(). Interact means to intersect over area, but try the combinatoral explosion of all other dimensions, i.e. what would make most sense when 2 stocks interact in a predator-prey relationship.

Additional variables will be prefixed with *prefix*.

stock_prepend

`stock_prepend(stock, param_call, name_part = NULL)`

Converts a g3_param or g3_param_table call, prefixing the parameter name with the stock name, and renaming any references to stock variables. If *name_part* given, will only add given part(s) of the stock name.

Returns *param_call* with the additions made.

stock_hasdim

`stock_hasdim(stock, dim_name)`

Returns `TRUE` iff *dim_name* is present in *stock*.

This is mostly useful to replace code depending on whether a dimension has been used. When used as part of an if statement, the if statement and the unused branch will be optimised away.

Examples

```
### debug_label
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10)
prey_stock <- g3_stock('herring', 1:3) %>% g3s_age(1,3)
g3_step(~debug_trace("Zero ", stock, "-", prey_stock, " biomass-consuming counter"))

### stock_assert
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10)
g3_step(~stock_assert(stock_with(stock, all(is.finite(stock__num))), stock, "__num became NaN/Inf"))

### stock_reshape
s <- g3_stock('s', seq(3, 21, 3))
s__num <- g3_stock_instance(s, 100)
agg <- g3_stock('agg', c(3, 10, 21), open_ended = FALSE)
g3_eval(~stock_iterate(s, stock_reshape(agg, stock_ss(s__num))))

### stock_ss
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10) %>% g3s_livesonareas(1)
stock__num <- g3_stock_instance(stock)
g3_step(~stock_iterate(stock, { x <- x + stock_ss(stock__num) }))
g3_step(~stock_ss(stock__num, area = 5))
# Lengthgroups for age_idx + 1
g3_step(~stock_ss(stock__num, age = default + 1))
# Vector for the entirety of the "next" area
g3_step(~stock_ss(stock__num, area = default + 1, vec = area))
g3_step(~stock_ss(stock__num, area = , age = j))

### stock_ssinv
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10) %>% g3s_livesonareas(1)
g3_step(~g3_step(~stock_ssinv(stock, 'age')))
g3_step(~g3_step(~stock_ssinv(stock, 'area')))

### stock_switch
stock <- g3_stock('halibut', 1:10) ; fleet_stock <- g3_fleet('igfs')
g3_step(~stock_switch(stock, halibut = 2, herring = 3, -1))
g3_step(~stock_switch(fleet_stock, halibut = 2, herring = 3, -1))
g3_step(~stock_switch(stock, halibut = stock__midlen, -1))

### stock_with
stock <- g3_stock('halibut', 1:10)
g3_step(~stock_with(stock, sum(stock__num)))

### stock_iterate
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10)
```

```
g3_step(~stock_iterate(stock, x <- x + stock_ss(stock__num)))

### stock_intersect
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10)
prey_stock <- g3_stock('herring', 1:3) %>% g3s_age(1,3)
g3_step(~stock_iterate(stock, stock_intersect(prey_stock, {
  x <- x + stock_ss(stock__num) + stock_ss(prey_stock__num)
})))

### stock_interact
stock <- g3_stock('halibut', 1:10) %>% g3s_age(1,10)
prey_stock <- g3_stock('herring', 1:3) %>% g3s_age(1,3)
g3_step(~stock_iterate(stock, stock_interact(prey_stock, {
  x <- x + stock_ss(stock__num) + stock_ss(prey_stock__num)
}, prefix = "prey" )))
```

---

stock *Gadget3 stock storage*

Description

Define multi-dimensional storage for use in models, mostly to contain state about stocks.

Usage

```
g3_stock(var_name, lengthgroups, open_ended = TRUE)

g3_stock_instance(stock, init_value = NA, desc = "")

g3_fleet(var_name)

g3_stock_def(stock, name)

g3s_clone(inner_stock, var_name)

g3_is_stock(stock)
```

Arguments

var_name	Prefix used for all instance variables of this stock. Can have multiple parts that will be concatentated together, see example.
lengthgroups	Vector defining length groups, each entry defining the minimum value.
open_ended	If TRUE, final *lengthgroups* value defines a group `x:Inf`. If FALSE, final *lengthgroups* value is the upper bound for the previous group.
inner_stock	A g3_stock or g3_fleet object to clone.
stock	A g3_stock or g3_fleet. For g3_stock_def, can also be a list of stock objects.

`init_value`	Intially the array will be filled with this constant, e.g. `1`, `0` or `NaN`
`desc`	Description of the array that will be included in models
`name`	Name of definition to extract, e.g. `"minlen"`.

Value

g3_stock: A `g3_stock` with length groups

g3_stock_instance: An array with dimensions matching the stock.

g3_fleet: A `g3_stock` without length groups

g3_stock_def: The definition of the given variable in the stock. If *stock* is a list, then a list with the definition of each will be returned.

g3s_clone: A `g3_stock` with identical dimensions to *inner_stock* but with a new name.

g3_is_stock: `TRUE` iff *stock* is a `g3_stock` object.

Examples

```
# Define a stock with 3 lengthgroups
stock <- g3_stock('name', c(1, 10, 100))

# Define a stock with a multi-part name. We can then dig out species name
stock <- g3_stock(c(species = 'ling', 'imm'), c(1, 10, 100))
stopifnot( stock$name == 'ling_imm' )
stopifnot( stock$name_parts[['species']] == 'ling' )

# Use stock_instance define storage for mean weight of stock,
# has dimensions matching what was defined above.
g3_stock_instance(stock, 1, "Mean weight")

# Can get definitions for multiple stocks in one go
stocks <- list(
  imm = g3_stock(c('st', 'imm'), 1:10),
  mat = g3_stock(c('st', 'mat'), 1:10) )
g3_stock_def(stocks, 'minlen')

# Retrieve the upperlen for the stock
g3_stock_def(stock, 'upperlen')

# Define a stock, not-open-ended. Now only 2 groups long
stock <- g3_stock('name', c(1, 10, 100), open_ended = FALSE)

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)

# Fleets don't have lengthgroups
stock <- g3_fleet('name') %>% g3s_livesonareas(1)
```

```
# Use stock_instance to see what the array would look like
g3_stock_instance(stock)
```

---

stock_age *Gadget3 stock age dimensions*

Description

Add age dimensions to g3_stock classes

Usage

```
g3s_age(inner_stock, minage, maxage)

g3s_agegroup(inner_stock, agegroups)
```

Arguments

inner_stock	A g3_stock that we extend with an age dimension
minage	Minimum age to store, integer.
maxage	Maximum age to store, integer.
agegroups	(optionally named) list of vectors of ages, grouping them together.

Value

g3s_age: A g3_stock with an additional 'age' dimension.
When iterating over the stock, iterate over each age in turn, *age* will be set to the current integer age.
When intersecting with another stock, only do anything if *age* is betweem *minage* and *maxage*.
If an age dimension already exists, it is redefined with new parameters.

g3s_agegroup: A g3_stock with an additional 'age' dimension.
When iterating over the stock, iterate over each agegroup in turn, *age* will be set to the first age in the group.
When intersecting with another stock, only do anything if *age* is part of one of the groups.

Examples

```
# Define a stock with 3 lengthgroups and 3 ages
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_age(5, 10)

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)
```

```
# Define a stock that groups age into "young" and "old"
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_agegroup(list(
        young = 5:7,
        old = 8:10))

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)
```

---

stock_areas *Gadget3 stock area dimensions*

Description

Add area dimensions to g3_stock classes

Usage

```
g3_areas(area_names)

g3s_livesonareas(inner_stock, areas)

g3s_areagroup(inner_stock, areagroups)
```

Arguments

area_names	A character vector of area names to use in the model
inner_stock	A g3_stock that we extend with an area dimension
areas	A vector of numeric areas that the stock is part of
areagroups	A list mapping names to vectors of numeric areas the stock is part of

Details

g3s_livesonareas breaks up a stock by area. Within a model, areas are only referred to by integer, however if these are named then that name will be used for report output.

Each area will be defined as a variable in your model as area_x, allowing you to use names in formulas, e.g. run_f = quote(area == area_x).

g3_areas is a helper to map a set of names to an integer

Inside a model each area will only be referred to by integer.

g3s_areagroup allows areas to be combined, this is mostly used internally by g3l_catchdistribution.

Value

g3_areas: A named integer vector, assigning each of *area_names* a number.

g3s_livesonareas: A g3_stock with an additional 'area' dimension.
When iterating over the stock, iterate over each area in turn, *area* will be set to the current integer area.
When intersecting with another stock, only do anything if *area* is also part of our list of areas.

g3s_areagroup: A g3_stock with an additional 'area' dimension.
When iterating over the stock, iterate over each areagroup in turn, *area* will be set to the first area in the group.
When intersecting with another stock, only do anything if *area* is part of one of the groups.

Examples

```
# Make a lookup so we can refer to areas by name
area_names <- g3_areas(c('a', 'b', 'c', 'd', 'e'))
stopifnot(area_names == c(a=1, b=2, c=3, d=4, e=5))

# Define a stock with 3 lengthgroups and 3 areas
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_livesonareas(area_names[c('a', 'b', 'c')])

# Area variables will be defined, so you can refer to them in formulas:
g3a_migrate(stock, g3_parameterized("migrate_spring"),
    run_f = ~area == area_b && cur_step == 2)

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)

# Define a stock that groups areas into "north" and "south"
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_areagroup(list(
        north = area_names[c('a', 'b', 'c')],
        south = area_names[c('d', 'e')]))

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)
```

---

stock_tag *Gadget3 tag dimension*

Description

Add tag dimensions to g3_stock classes

Usage

```
g3s_tag(inner_stock, tag_ids, force_untagged = TRUE)
```

Arguments

inner_stock	A g3_stock that we extend with an area dimension
tag_ids	A vector of numeric tags the stock can have, generated by seq_along, e.g. Tag ID 0 is considered to be "untagged".
force_untagged	If TRUE, if "untagged" tag 0 isn't present it will be added.

Value

g3s_tag: A g3_stock with an additional 'tag' dimension.

When iterating over the stock, iterate over each tag in turn, *tag* will be set to the current integer area.

When interacting with another stock, iterate over each tag in turn, the variable name will depend on the scenario, e.g. *prey_tag*.

Examples

```
# Make a lookup of text names to integers
tags <- c('H1-00', 'H1-01')
tags <- structure(seq_along(tags), names = tags)

# prey_a can have any of these tags
prey_a <- g3_stock('prey_a', seq(1, 10)) %>% g3s_tag(tags)

# Use stock_instance to see what the array would look like
g3_stock_instance(prey_a)
```

---

stock_time *Gadget3 stock time dimensions*

Description

Add time dimensions to g3_stock classes

Usage

```
g3s_time_convert(year_or_time, step = NULL)

g3s_time(inner_stock, times, year = NULL, step = NULL)
```

Arguments

year_or_time	Etiher vector of years, or vector of year & step strings, e.g. "1999-01".
year	Vector of years, used to generate *times* if provided.
step	Vector of steps, used to generate *times* if provided.
inner_stock	A g3_stock that we extend with a time dimension
times	A vector of year/step integers as generated by *g3s_time_convert*

Value

g3s_time_convert: A single integer vector representing *year* and *step*.
If *step* is NULL, returns *year*, otherwise *year* * 1000 + *step*.

g3s_time: A g3_stock with an additional 'time' dimension.
If *year*/*step* provided, time is defined by those, otherwise *times*.
The g3_stock will not support iterating, only intersecting.
When intersecting with another stock, only do anything if *cur_year* and *cur_step* matches a time stored in the vector

Examples

```
# Define a stock with 3 lengthgroups and 3 years, not continuous
# When used, all steps within a year will be aggregated, year 2002 will be ignored.
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_time(year = c(2000, 2001, 2003))

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)

# Define a stock with 3 lengthgroups and 3 years, 2 steps
# The dimension will have 6 entries, 2000.1, 2000.2, 2001.1, 2001.2, 2002.1, 2002.2
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_time(year = c(2000, 2001, 2002), step = 1:2)

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)

# g3s_time_convert is best used with a data.frame
data <- read.table(header = TRUE, text = '
year step
2001 1
2001 2
# NB: No "2002 1"
2002 2
')
stock <- g3_stock('name', c(1, 10, 100)) %>%
    g3s_time(times = g3s_time_convert(data$year, data$step))

# Will also parse strings
```

```
g3s_time_convert(c("1999-01", "1999-02"))

# Use stock_instance to see what the array would look like
g3_stock_instance(stock)
```

---

suitability *Gadget3 suitability formulae*

Description

Formula-returning functions describing length suitability relationships.

Usage

```
g3_suitability_exponentiall50(
   alpha = g3_parameterized("alpha", by_stock = by_stock, by_predator = by_predator),
   l50 = g3_parameterized("l50", by_stock = by_stock, by_predator = by_predator),
    by_stock = TRUE,
    by_predator = TRUE)

g3_suitability_andersen(p0, p1, p2, p3 = p4, p4, p5 = quote( predator_length ))

g3_suitability_andersenfleet(
        p0 = g3_parameterized('andersen.p0', value = 0, optimise = FALSE,
                              by_stock = by_stock),
        p1 = g3_parameterized('andersen.p1', value = log(2),
                              by_stock = by_stock, by_predator = by_predator),
        p2 = g3_parameterized('andersen.p2', value = 1, optimise = FALSE,
                              by_stock = by_stock),
     p3 = g3_parameterized('andersen.p3', value = 0.1, exponentiate = exponentiate,
                              by_stock = by_stock, by_predator = by_predator),
     p4 = g3_parameterized('andersen.p4', value = 0.1, exponentiate = exponentiate,
                              by_stock = by_stock, by_predator = by_predator),
        p5 = quote( stock__maxmidlen ),
        by_stock = TRUE,
        by_predator = TRUE,
        exponentiate = TRUE)

g3_suitability_gamma(alpha, beta, gamma)

g3_suitability_exponential(alpha, beta, gamma, delta)

g3_suitability_straightline(alpha, beta)

g3_suitability_constant(
   suit = g3_parameterized("suit", by_stock = by_stock, by_predator = by_predator),
    by_stock = TRUE,
```

```
    by_predator = TRUE )

g3_suitability_richards(p0, p1, p2, p3, p4)
```

Arguments

`suit`, `alpha`, `beta`, `gamma`, `delta`, `l50`, `p0`, `p1`, `p2`, `p3`, `p4`, `p5`	formula substituted into calcuations, see below.
`by_stock`	Change the default parameterisation (e.g. to be by 'species'), passed through to default calls to `g3_parameterized`.
`by_predator`	Change the default parameterisation (e.g. to be by 'fleet'), passed through to default calls to `g3_parameterized`.
`exponentiate`	Exponentiate parameters,passed through to default calls to `g3_parameterized`.

Details

When using these to describe a predator/prey relationship, the stock midlength l will refer to the prey midlength.

Value

All functions return a formula for use in `g3a_predate_fleet`'s *suitabilities* argument:

g3_suitability_exponentiall50: A logarithmic dependence on the length of the prey as given by the following equation (note that the prey length dependence is actually dependant on the difference between the length of the prey and l_{50}):

$$\frac{1}{1 + e^{-\alpha(l - l_{50})}}$$

l Vector of stock midlength for each lengthgroup

l_{50} Length of the stock with a 50% probability of predation, from parameter *l50*

g3_suitability_andersen: This is a more general suitability function that is dependant on the ratio of the predator length to the prey length as given by the following equation:

If $p_3 = p_4$:

$$p_0 + p_2 e^{-\frac{(x - p_1)^2}{p_4}}$$

Otherwise:

$$p_0 + p_2 e^{-\frac{(x - p_1)^2}{p_4}} * \min(\max(p_1 - x, 0), 1) + p_2 e^{-\frac{(x - p_1)^2}{p_3}} * \min(\max(x, 0), 1)$$

...i.e if $\log \frac{L}{l} <= p_1$ then p_3 used in place of p_4.

x $\log \frac{p_5}{l}$

L Vector of predator midlength for each lengthgroup

l Vector of stock midlength for each lengthgroup

p_0 .. p_4 Function parameter *p0* .. *p4*

p_5 Function parameter *p5*, if unspecified uses L, Vector of predator midlength for each lengthgroup.

NB: Specifying p_5 is equivalent to using the `andersenfleet` function in gadget2.

g3_suitability_andersenfleet: A simplified version of `g3_suitability_andersen`, suitable for predation by fleets, as the defaults do not rely on the predator's length.

g3_suitability_gamma: This is a suitability function that is more suitable for use when considering the predation by a fleet, where the parameter γ would represent the size of the mesh used by the fleet (specified in centimetres).

$$(\frac{l}{(\alpha-1)\beta\gamma})^{(\alpha-1)e^{\alpha-1-\frac{l}{\beta\gamma}}}$$

l Vector of stock midlength for each lengthgroup

α Function parameter *alpha*

β Function parameter *beta*

γ Function parameter *gamma*

This is a suitability function that is more suitable for use when considering the predation by a fleet, where the parameter γ would represent the size of the mesh used by the fleet (specified in centimetres).

g3_suitability_exponential: This is a suitability function that has a logarithmic dependence on both the length of the predator and the length of the prey as given by the following equation:

$$\frac{\delta}{1+e^{-\alpha-\beta l-\gamma L}}$$

L Vector of predator midlength for each lengthgroup

l Vector of stock midlength for each lengthgroup

α Function parameter *alpha*

β Function parameter *beta*

γ Function parameter *gamma*

δ Function parameter *delta*

g3_suitability_straightline: Returns a formula for use in predation function's *suitabilities* argument:

$$\alpha+\beta l$$

l Vector of stock midlength for each lengthgroup

α Function parameter *alpha*

β Function parameter *beta*

g3_suitability_constant: Returns a formula for use in predation function's *suitabilities* argument:

$$\alpha$$

α Function parameter *suit*, i.e. the "prey.predator.suit" model parameter by default

g3_suitability_richards: Returns a formula for use in predation function's *suitabilities* argument:

$$\left(\frac{p_3}{1+e^{-p_0-p_1 l-p_2 L}}\right)^{\frac{1}{p_4}}$$

L Vector of predator midlength for each lengthgroup

l Vector of stock midlength for each lengthgroup

p_0 .. p_4 Function parameter *p0* .. *p4*

This is an extension to g3_suitability_exponential.

See Also

https://gadget-framework.github.io/gadget2/userguide/chap-stock.html#sec-suitability,

Examples

```
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock(c(species = 'ling', 'mat'), seq(20, 156, 4)) %>% g3s_age(5, 15)
igfs <- g3_fleet('igfs')

igfs_landings <-
  structure(expand.grid(year=1990:1994, step=2, area=1, total_weight=1),
            area_group = list(`1` = 1))

# Generate a fleet predation action using g3_suitability_exponentiall50
predate_action <- g3a_predate_fleet(
    igfs,
    list(ling_imm, ling_mat),
    suitabilities = list(
        ling_imm = g3_suitability_exponentiall50(
            g3_parameterized('lln.alpha', by_stock = 'species'),
            g3_parameterized('lln.l50', by_stock = 'species')),
        ling_mat = g3_suitability_exponentiall50(
            g3_parameterized('lln.alpha', by_stock = 'species'),
            g3_parameterized('lln.l50', by_stock = 'species'))),
    catchability = g3a_predate_catchability_totalfleet(
        g3_timeareadata('igfs_landings', igfs_landings)))

# You can use g3_eval to directly calculate values for a stock:
g3_eval(
    g3_suitability_exponentiall50(alpha = 0.2, l50 = 60),
    stock = g3_stock('x', seq(0, 100, 10)) )

## Plots
suit_plot <- function (
    suit_f,
    stock = g3_stock('x', seq(0, 100, 5)),
    predator_length = 140,
    cols = rainbow(5) ) {
  par(mar = c(2,2,2,2), cex.main = 1)
```

```
  for (a in seq_along(cols)) curve(
    g3_eval(
      suit_f,
      a = a,
      stock = stock,
      predator_length = predator_length )[x %/% g3_stock_def(stock, 'dl')[[1]] + 1],
    from = min(g3_stock_def(stock, 'minlen')),
    to = max(g3_stock_def(stock, 'minlen')),

    col = cols[[a]],
    main = deparse1(sys.call()[[2]]), xlab = "", ylab = "",
    add = (a != 1) )
}

suit_plot(g3_suitability_exponentiall50(alpha = ~a * 0.1, l50 = 50))
suit_plot(g3_suitability_andersen(0, log(2), 1, p3 = ~a * 0.1, 0.1, 140))
suit_plot(g3_suitability_andersen(0, log(2), 1, 0.1, p4 = ~a * 0.1, 140))
suit_plot(g3_suitability_gamma(alpha = ~2 + a * 0.1, beta = 1, gamma = 40))
suit_plot(g3_suitability_exponential(0, ~0.01 * a, 0, 1))
suit_plot(g3_suitability_straightline(alpha = 0.1, beta = ~0.01 * a))
suit_plot(g3_suitability_constant(~a * 0.1))
suit_plot(g3_suitability_richards(0, 0.05, 0, 1, ~0.1 * a))
```

`timedata` *Gadget3 time-based data*

Description

Convert time-based data into a formula to lookup values

Usage

```
g3_timeareadata(lookup_name, df, value_field = "total_weight", areas = NULL)
```

Arguments

`lookup_name`	A unique name for this lookup, e.g. `"igfs_landings"`.
`df`	A data.frame with any of columns out of age, area, year and step, finally *value_field*.
`value_field`	Column name that contains output value.
`areas`	Named integer vector of area names to integer values. See `g3s_livesonareas`.

Value

A formula object that looks up *value_field* for the current values of `age`, `area`, `cur_year` and `cur_step`, depending on the columns in *df*. If there's no match, return 0.

Examples

```
ling_imm <- g3_stock(c(species = 'ling', 'imm'), seq(20, 156, 4)) %>% g3s_age(3, 10)
ling_mat <- g3_stock(c(species = 'ling', 'mat'), seq(20, 156, 4)) %>% g3s_age(5, 15)
igfs <- g3_fleet('igfs')

igfs_landings <-
  structure(expand.grid(year=1990:1994, step=2, area=1, total_weight=1),
            area_group = list(`1` = 1))

# Generate a fleet predation action, use g3_timeareadata to supply landings
# NB: Since igfs_landings only contains values for step=2, there will be no
#     predation on other steps (since g3_timeareadata will return 0).
predate_action <- g3a_predate_fleet(
    igfs,
    list(ling_imm, ling_mat),
    suitabilities = list(
        ling_imm = g3_suitability_exponentiall50(
            g3_parameterized('lln.alpha', by_stock = 'species'),
            g3_parameterized('lln.l50', by_stock = 'species')),
        ling_mat = g3_suitability_exponentiall50(
            g3_parameterized('lln.alpha', by_stock = 'species'),
            g3_parameterized('lln.l50', by_stock = 'species'))),
    catchability = g3a_predate_catchability_totalfleet(
        g3_timeareadata('igfs_landings', igfs_landings)))
```

---

timevariable *Gadget3 time-based formulas*

Description

Switch formula based on current time step

Usage

```
g3_timevariable(lookup_name, fs)
```

Arguments

lookup_name	A unique name for this lookup, e.g. "igfs_landings".
fs	A list of formula objects, named with either "init", "(year)" or "(year)-(step)". When the matching time step is reached, the value of the lookup will be changed.

Details

This is mostly for backwards compatibility with gadget2, before using this, consider other simpler options, e.g. g3_timeareadata or the *by_year* option in g3_parameterized.

Value

A formula object that will switch values at the given time points.

Examples

```
ling_imm <- g3_stock('ling_imm', seq(20, 156, 4)) %>% g3s_age(3, 10)

naturalmortality_action <- g3a_naturalmortality(ling_imm,
    g3a_naturalmortality_exp( g3_timevariable("lingimm.M", list(
        # Start off using lingimm.M.early
        "init" = g3_parameterized("lingimm.M.early"),
        # At 2005 step 2, switch to lingimm.M.mid
        "2005-02" = g3_parameterized("lingimm.M.mid"),
        # At 2010 step 1, switch to lingimm.M.late
        "2010" = g3_parameterized("lingimm.M.late")))))
```

Index