Package ‘MixFishSim’

March 16, 2018

Title Mixed Fishery fleet dynamics simulation tool

Version 0.0.0.9000

Description A simulation framework for evaluating fleet dynamics in mixed fisheries.

Depends R (>= 3.3.1),

Imports CircStats,
doParallel,
dplyr,
parallel,
RandomFields,
Rcpp

License What license is it under?

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat,
ggplot2,
dplyr,
akima

LinkingTo Rcpp

R topics documented:

`baranov_f` *Baranov F*

Description

`baranov_f` provides the function to solve in `find_f` for estimating weekly fishing mortality from catch (C), biomass (B) and natural mortality (M). It's based on the standard Baranov catch equation.

Usage

```
baranov_f(F, C, B, M)
```

Arguments

`F`	is the fishing mortality rate to solve.
`C`	is a Numeric vector detailing the catch at wk_t
`B`	is a Numeric vector of the biomass at wk_t
`M`	is a Numeric vector of the natural mortality rate at wk_t

Value

returns nothing, is objective to be solved by `find_f`

Examples

```
## No examples
```

`close_areas` *Close areas*

Description

The `close_areas` function implements the closures according to the settings from init_closure and passes the areas to go_fish. Its an internal function, requiring no user input.

Usage

```
close_areas(sim_init = sim_init, closure_init = NULL,
  commercial_logs = NULL, survey_logs = NULL, real_pop = NULL, t = t)
```

Arguments

`closure_init`	is the output from init_clousre.
`commercial_logs`	is the commercial landings data, the output from combine_logs. Only needed if closure 'basis' = 'commercial'.
`survey_logs`	is the survey data, the survey[["log.mat"]]. Only needed if closure 'basis' is 'survey'.
`real_pop`	is the popualtions as recorded. Only needed if closure 'basis' is 'real_pop'.

Value

is a list of closed cells, to pass to go_fish

Examples

```
None
```

combine_logs *Combine logs*

Description

`combine_logs` is a helper function to convert the list of fleet and vessels catch logs into a single dataframe.

Usage

```
combine_logs(fleets_catches)
```

Arguments

`fleets_catches` is the list output of fleets_catches from run_sim

Value

is a dataframe of the fleet and vessel catches in logbook format

Examples

```
logs <- combine_logs(fleets_catches)
## Not run
```

create_fields *Create species distribution fields*

Description

`create_fields` parametrises and returns the spatio-temporal fields used for the spatial distribution of fish populations and movement in space and time for the simulations.

The spatio-temporal fields are generated using `spate.sim` function from the *spate* package using an advective-diffusion Stochastic Partial Differential Equation (SPDE). See *Lindgren 2011 and Sigrist 2015* for further detail.

Usage

```
create_fields(npt = 1000, t = 1, seed = 123, n.spp = NULL,
  spp.ctrl = NULL, plot.dist = FALSE, plot.file = getwd())
```

Arguments

- `npt` Numeric integer with the dimensions of the field in *npt * npt*
- `t` Numeric integer with the number of time-steps in the simulation
- `seed` (Optional) Numeric integer with the seed for the simulation
- `n.spp` Numeric integer with the number of species to be simulated. Each species must have an individual control list as detailed below.
- `spp.ctrl` List of controls to generate each species spatio-temporal distribution. Must be of the form spp.ctrl = list(spp.1 = c(rho0 = 0.001, ...), spp.2 = c(rho0 = 0.001, ..),..) and contain the following:
  - **rho0** (>=*0*) Controls the range in a matern covariance structure.
  - **sigma2** (>=*0*) Controls the marginal variance (i.e. process error) in the matern (>=*0*) covariance structure.
  - **zeta** (>=*0*) Damping parameter; regulates the temporal correlation.
  - **rho1** (>=*0*) Range parameter for the diffusion process
  - **gamma** (>=*0*) Controls the level of anisotropy
  - **alpha** (*[0,* $\pi/2$*]*) Controls the direction of anisotropy
  - **muX** (*[-0.5, 0.5]*) x component of drift effect
  - **muY** (*[-0.5, 0.5]*) y component of drift effect
  - **tau2** (>=*0*) Nugget effect (measurement error)
  - **nu** Smoothness parameter for the matern covariance function
- `plot.dist` Boolean, whether to plot the distributions to file
- `plot.file` path to save the plots of the species distributions

Value

Silently returns a list of spatial distributions with first level of the list being the population (1 -> n.spp) and the second being time (1 -> t). If `plot.dist = TRUE` it produces an image of the spatial distributions at each time step for each of the populations saved to the working directory (unless specified otherwise in `plot.file`)

Examples

```
fields <- create_fields(n.spp = 1, t = 2,
      spp.ctrl = list(
      'spp.1' = c('rho0' = 0.1, 'sigma2' = 1, 'zeta' = 0.1,
  'rho1' = 0.01, 'gamma' = 0.3, 'alpha' = pi/4,
  'muX' = -0.05, 'muY' = -0.05, 'tau2' = 0, 'nu' = 1.5)),
plot.dist = TRUE, plot.file = getwd())
```

create_hab *Create habitat distribution fields*

Description

create_hab parametrises and returns the spatial fields used for the distribution of suitable habitat for the populations in the simulation.

The spatial fields are generated using RFsimulate function from the *RandomFields* package.

Usage

```
create_hab(sim_init = sim, seed = 123, spp.ctrl = NULL,
  spawn_areas = NULL, spwn_mult = 10, plot.dist = FALSE,
  plot.file = getwd(), cores = 3)
```

Arguments

spp.ctrl	List of controls to generate suitable habitat for each species. Must be of the form spp.ctrl = list(spp.1 = c(var = 20, ...), spp.2 = c(var = 10, ..),..) and contain the following: • **nu** (*>=0*) • **var** (*>=0*) Controls the range in a matern covariance • **scale** (*>=0*) • **Aniso** (*matrix, dim = c(2,2)*)
plot.dist	Boolean, whether to plot the distributions to file
plot.file	path to save the plots of the species distributions
sim	is the parameter settings for the simulation, made by init_sim function.

Value

Silently returns a list of spatial distributions of suitable habitat with first level of the list being the population (1 -> n.spp). If plot.dist = TRUE it produces an image of the spatial distributions at each time step for each of the populations saved to the working directory (unless specified otherwise in plot.file)

Examples

```
hab <- create_hab(sim.init = sim.init, spp.ctrl = list(
      'spp.1' = list('nu' = 1/0.15, var = 1, scale = 10, Aniso =
      matrix(nc=2, c(1.5, 3, -3, 4)))), spawn_areas = list("spp1" =
      list("area1" = c(2,4,6,8))), list("spp2" = list("area1" =
      c(0,10,23,35))), spwn_mult = 10, plot.dist = TRUE, plot.file =   getwd())
```

`create_spawn_hab` *create spawning habitat*

Description

`create_spawn_hab` modifies the habitat preference maps created by `create_hab` to account for spawning habitat preference - can be used as a substitute during spawning periods.

Usage

```
create_spawn_hab(hab = hab, spwnareas = NULL, mult = 10)
```

Arguments

`hab`	is the habitat preference for the population
`spwnareas`	is a list of Numeric vectors with the West, East, South and North dimensions of the spawning areas, in the form list(spwn1 = c(x1, x2, y1, y2)
`mult`	is a Numeric with the attractiveness of the spawning area (a multiplier)

Value

is the new habitat preference, taking account of the spawning area

Examples

```
create_spawn_hab(hab = matrix(nc = 100, runif(100 *
100)), spwnareas = list(spwn1 = c(20, 30, 50, 60)), mult = 10)
```

`define_spawn` *define spawning areas*

Description

`define_spawn` is an auxiliary function called by `create_spawn_hab` to create the spawning habitat preferences.

Usage

```
define_spawn(coord = NULL, spwn = NULL, mult = 10)
```

Arguments

`coord`	is a List of Numeric vectors of the boundaries of the spawning areas, i.e. list(spwn1 = c(x1, x2, y1, y2), spwn2 = ...)
`spwn`	is a Numeric matrix of 1s fed in by `create_spawn_hab`
`mult`	is a Numeric of the attractiveness of the spawning areas

Value

a matrix of spawning preference

Examples

```
define_spawn(coord = list(spwn1 = c(2,4,2,4)), spwn = matrix(nc = 3, runif(9)), mult = 10)
```

---

deg2rad *Degrees to radians*

Description

`deg2rad` is a helper function to covert decimal degrees to radians

Usage

```
deg2rad(d)
```

Arguments

`r`	is the bearing in radians

Value

is the bearing in degrees

Examples

```
deg2rad(90)
```

---

delay_diff *Delay-difference (weekly)*

Description

`delay_diff` implements a two-stage delay-difference model with a weekly time-step after *Dichmont 2003*. Given the starting biomass, overall mortality and recruitment it returns the biomass in wk+1.

Usage

```
delay_diff(K = 0.3, F = NULL, M = 0.2, wt = 1, wtm1 = 0.1, R = NULL,
  B = NULL, Bm1 = NULL, al = NULL, alm1 = NULL)
```

Arguments

K	is a Numeric vector describing growth. Note: K is transformed to rho with $\rho = exp{-K}$ for the model. estimate of instantaneous fishing mortality (obtained elsewhere, via find_f and baranov_f functions.
F	is the weekly fishing mortality rate.
M	is a Numeric vector of the instantaneous rate of natural mortality for the population
wt	is a Numeric vector of the weight of a fish when fully recruited
wtm1	is a Numeric vector of the weight of a fish before its recruited
R	is a Numeric vector of the annual recruitment for the population in numbers
B	is the biomass of the population during wk_t
Bm1	is a Numeric vector of the biomass of the population in the previous week wk_{t-1}
al	is a Numeric vector of the proportion of recruits to the fishery in wk_t
alm1	is a Numeric vector of the proportion of recruits to the fishery in wk_{t-1}

Value

Returns the biomass at the beginning of the following week, wk_{t+1}

Examples

```
delay_diff(K = 0.3, F = 0.2, M = 0.2, wt = 1, wtm1 = 0.1, R = 1e6, B = 1e5,
Bm1 = 1e4, al = 0.5, alm1 = 0.1)
```

---

distance_calc *distance calculation*

Description

distance_calc calculates the euclidean distance between two cell references.

Usage

```
distance_calc(x1, y1, x2, y2)
```

Arguments

x1	is an integar for the starting x position
y1	is an integar for the starting y position
x2	is an integar for the end x position
y2	is an integar for the end y position

Value

is a distance between the two cells

Examples

```
distance_calc(2, 3, 5, 7)
```

---

find_f *find F (fishing mortality)*

Description

find_f uses uniroot to find the fishing mortality rate given the catch, biomass and natural mortality using the baranov_f objective function.

Usage

```
find_f(C = C, B = B, M = M, FUN = baronov_f)
```

Arguments

C	is a Numeric vector detailing the catch at wk_t
B	is a Numeric vector of the biomass at wk_t
M	is a Numeic vector of the natural mortality rate at wk_t
FUN	is the objective function, here the Baranov equation baranov_f

Value

Gives the fishing mortality estimate *F*

Examples

```
find_f(C = 3000, B = 12000, M = 0.2, FUN = baranov_f)
```

---

find_spat_f *find spatial Fs (fishing mortality rates)*

Description

find_spat_f uses uniroot to find the fishing mortality rate for a population given the catch, biomass and natural mortality using the baranov_f objective function.

Usage

```
find_spat_f(sim_init = NULL, C = C, B = B, M = M, FUN = baranov_f)
```

Arguments

`sim_init`	is the parameterised sim settings, made by `init_sim`
`C`	is a Numeric vector detailing the catch at wk_t
`B`	is a Numeric vector of the biomass at wk_t
`M`	is a Numeic vector of the natural mortality rate at wk_t
`FUN`	is the objective function, here the Baranov equation baranov_f

Value

Gives a matrix the spatial fishing mortality estimate *F*

Examples

```
find_spat_f(sim_init = sim, C = matrix(1000,3000, nc =2), B =
matrix(12000,10000, ncol = 2), M = 0.2, FUN = baranov_f)
```

`find_spat_f_pops` *find spatial f pops*

Description

`find_spat_f_pops` applies the `find_spat_f` function to all the populations, returning the spatial fishing mortality rates for each of the populations.

Usage

```
find_spat_f_pops(FUN = find_spat_f, sim_init = sim, C = C, B = B,
  dem_params = NULL, ...)
```

Arguments

`FUN`	is the `find_spat_f` function
`sim_init`	is the simulation settings initialised by `init_sim`
`C`	is the spatial catch matrices for all populations
`B`	is the spatial biomass for all populations
`dem_params`	are the demographic parameters for all populations (containing the natural mortality rate, M.

Examples

```
None as yet
```

`get_bearing` *Get bearing function*

Description

`get_bearing` is a function to calculate a new bearing for a vessel. The new bearing is determined from the Von Mises circular distribution, with a concentration parameter, *k* which is linked to the value of the recent tow. Thus, if a vessel has a good tow, its more likely to turn round and fish again in the same area.

Usage

```
get_bearing(b = NULL, k = NULL)
```

Arguments

`b`	is a Numeric based on decimal degrees (0 - 360) of the current bearing for the vessel
`k`	is a Numeric [0-100] for the concentration parameter determining the likely new direction for the vessel.

Value

bearing - is the new bearing for the vessel

Examples

```
get_bearing(b = 270, k = 100)
```

`go_fish` *Go fish*

Description

`go_fish` is a function used to apply the fishing simulation model

Usage

```
go_fish(sim_init = NULL, fleet_params = NULL, fleet_catches = NULL,
  sp_fleet_catches = NULL, pops = NULL, closed_areas = NULL, t = t)
```

Arguments

`sim_init` is the initialised object from `init_sim`.

`fleet_params` is the parameter settings initialised from `_init_fleets`

`fleet_catches` is the DF initialised from `_init_fleets`

`sp_fleet_catches` is a list of spatial catches (as a Numeric matrix) for the fleet of each population @param closed_areas is a dataframe with the x,y coordinates are any closed areas, provided internally by close_areas

Value

is a list containing i) the fleet catch dataframes , ii) the spatial catches of each population

---

`go_fish_fleet` *Go fish fleet*

Description

`go_fish_fleet` applies the function `go_fish` to the entire fleet with an lapply.

Usage

```
go_fish_fleet(FUN = go_fish, sim_init = NULL, fleets_params = NULL,
  fleets_catches = NULL, sp_fleets_catches = NULL, pops = NULL,
  closed_areas = NULL, t = t, ...)
```

Arguments

`fleets_params` is the parameter settings initialised from `_init_fleets`

`fleets_catches` is the DF initialised from `_init_fleets`

`closed_areas` is a dataframe with the x,y coordinates are any closed

`Pop` is the population matrix for all populations

`sp_fleet_catches` is a list of spatial catches (as a Numeric matrix) for the fleet of each population

Value

is a list with the objects catch detailing the fleet catches and catch_matrices detailing the spatial catches, to input to the delay difference model

Examples

```
None as yet
```

init_closure *Initialise spatial closure(s)*

Description

`init_closure` sets up the parameters for spatial closure(s) in the simulation.

Usage

```
init_closure(input_coords = NULL, basis = "commercial",
  rationale = "high_pop", spp1 = "spp1", spp2, year_start = 1,
  year_basis = NULL, closure_thresh = 0.95, sc = 1, temp_dyn = "annual")
```

Arguments

`input_coords`	is a dataframe of x,y coordinates defining the closure(s). If the temp_dyn are not static, the list should be multilayered with the [[week/month]][x, y]
`basis`	is a character string detailing the data used to define a closure 'on the fly'. Can be *survey* to be based on survey data, *commercial* to be based on commercial data, *real_pop* to be based on the simulated population. Not needed if coordinates defined.
`rationale`	is the basis for any 'on the fly' closure. Can be *high_pop* for the areas of a highest population or *high_ratio* for the areas of the highest ratio of population 1: population 2. Not needed if coordinates defined.
`spp1`	is the first population as basis for the closure. If rationale = high_pop then that should go here If rationale = high_ratio, its the target (high quota) population. Not needed if coordinates are defined.
`spp2`	is the second population when rationale = high_ratio, the lowest quota population. Not needed if coordinates provided or rationale = high_pop.
`year_start`	is a Numeric indicating the first year the spatial closure(s) shoud be implemented.
`year_basis`	is a vector indicating the years of data the closure is based on...Must be before `year_start`. If `NULL` then closure will be calculated dynamically each year.
`closure_thresh`	is the quantile of catches or high catch ratio which determines closed cells
`sc`	is a Numeric indicating the scale of data to use for the closure, e.g. if the data is aggregated to 2 x 2 cells, is 2.
`temp_dyn`	is a character string detailing whether closures should be temporally 'annual', or change 'monthly' or 'weekly'.

Value

is a list of parameter settings for the spatial closures which serves as an input to run_sim.

Examples

```
Not as yet
```

`init_fleet` *Initialise fleet*

Description

`init_fleet` sets up the parameters and results data frame to record the catches from the simulation.

Usage

```
init_fleet(sim_init = NULL, VPT = NULL, Qs = NULL, step_params = NULL,
  past_knowledge = FALSE, past_year_month = FALSE, past_trip = FALSE,
  threshold = NULL)
```

Arguments

`sim_init`	is the output (a list) from the `sim_init` function with the indexing for the simulation.
`VPT`	is a named vector of numerics detailing the value-per-tonne for catches from each of the species (same for all fleets)
`Qs`	is a list (an element for each fleet) with each element containing a named vector with the catchability parameters for each species the vessels in the fleet
`step_params`	is a list (an element for each fleet) with each element containing a named vector with the step parameters used in `step_length`. This must include the named elements **rate**, **B1**, **B2**, **B3**.
`past_knowledge`	is a Boolean (TRUE / FALSE) whether past knowledge should determine fishing location (only after the first year)
`past_year_month`	is a Boolean (TRUE / FALSE) that indicates whether the same month in previous years should be included in the past knowledge decision
`past_trip`	is a Boolean (TRUE / FALSE) that indicates whether the past trip undertaken should be included in the past knowledge decision
`knowledge_threshold`	is a numeric (0 - 1) detailing the threshold at which a fishing tow should be considered "good" and included in the selection of possible choices of starting fishing locations in future tows.

Value

is a list with three elements containing i) the fleet parameters, a named list **fleet_params**, ii) the fleet catches, **catches_list**, which is a list of a list. For the**catches_list** the first element denotes the fleet number, the second element is the vessel number with a dataframe for recording the vessels catches. Finally, iii) is the spatial catches for the fleets, which is a list (fleet) containing a list (vessels) containing a list (population) - which is to be passed to the delay difference model.

Examples

```
None yet, to add
```

init_moveCov *Initialise movement covariates*

Description

This function creates a list of covariates, to be used

Usage

```
init_moveCov(sim_init = NULL, steps = 52, spp_tol = NULL)
```

Arguments

sim_init	is the output from the function init_sim.
steps	is a Numeric with the number of timesteps over which the covariate changes
spp_tol	is a named list (each species) with a list of mean (mu) and variance (va) for the normal distribution for thermal tolerance.

Examples

```
None
```

init_pop *Initialise populations*

Description

`init_pop` sets up the populations spatial distribution based on the habitat preference, starting cell and 'n' numbers of movements for all populations in the simulation.

Usage

```
init_pop(sim_init = sim_init, Bio = NULL, hab = NULL, start_cell = NULL,
  lambda = NULL, init_move_steps = 10, rec_params = NULL, rec_wk = NULL,
  spwn_wk = NULL, M = NULL, K = NULL, cores = 3)
```

Arguments

Bio	is a named Numeric vector of the starting (total) biomass for each of the populations.
hab	is the list of Matrices with the habitat preferences created by `create_hab`
start_cell	is a list of Numeric vectors with the starting cells for the populations
lambda	is the strength that the movement distance decays at in the `move_prob` function

`init_move_steps`
: is a Numeric indicating the number of movements to initialise for the population distributions

`rec_params`
: is a list with an element for each population, containing a vector of the stock recruit parameters which must contain **model**, **a**, **b** and **cv**. See `Recr` for details.

`rec_wk`
: is a list with an element for each population, containing a vector of the weeks in which recruitment takes place for the population

`spwn_wk`
: is a list with an element for each population, containing a vector of the weeks in which spawning takes place for the population

`M`
: is a named vector, with the annual natural mortality rate for each population

`K`
: is a named vector, with the annual growth rate for each population

`spawn_areas`
: is a list of lists, with the first level the population ("spp1" etc..) and the second the boundary coordinates (x1, x2, y1, y2) for the `create_spawn_hab` function

Value

The function returns the recording vectors at the population level, the spatial matrices for the starting population densities and the demographic parameters for each population

Examples

```
init_pop(sim_init = sim_init, Bio = c("spp1" = 1e6, "spp2" = 2e5), hab = list(spp1 = matrix(nc = 10,
runif(10*10)), spp2 = matrix(nc = 10, runif(10*10)), lambda = c("spp1" =
0.2, "spp2" = 0.3), init_move_steps = 10), rec_params = list("spp1" =
c("model" = "BH", "a" = 10, "b" = 50, "cv" = 0.2), "spp2" = c("model" = "BH",
"a" = 1, "b" = 8, "cv" = 0.2)), rec_wk = list("spp1" = 13:16, "spp2" =
13:18), spwn_wk = list("spp1" = 15:18, "spp2" = 18:20),M = c("spp1" = 0.2,
"spp2" = 0.1), K = c("spp1" = 0.3, "spp2" = 0.2))
Note, example will not have the right biomass
```

`init_sim` *Initialise simulation*

Description

`init_sim` sets up the general simulation parameters such as number of tows in a day, number of days fished in a week, how often species movement occurs and number of years for the simulation. It also creates some vector and matrix structures which are used in the `init_pop` and `init_fleet` functions.

Usage

```
init_sim(n_years = 1, n_tows_day = 4, n_days_wk_fished = 5,
  n_fleets = 1, n_vessels = 1, n_species = 1, nrows = nrows,
  ncols = ncols, move_freq = 2)
```

Arguments

`n_years`	is an integer defining the number of years for the simulation
`n_days_wk_fished`	is an integar defining the number of days in a calendar week that are fished (e.g. 5 (out of 7))
`n_fleets`	is an integar defining the number of fleets in the simulation
`n_vessels`	is an integar defining the number of vessels in each fleet
`n_species`	is an integar defining the number of species in the simulation
`nrows`	Numeric integer with the y dimension of the field in *nrow * ncol*
`ncols`	Numeric integer with the x dimension of the field in *nrow * ncol*
`move_freq`	is an integar defining the duration (in weeks) between spatial movements for the populations
`n_tow_day`	is an integar defining the number of tows in a days fishing

Value

is a list of lists, detailing the indexs and data formats necessary for the simulation.

Examples

```
init_sim(n_years = 1, n_tows_day = 4, n_days_wk_fished = 5,
n_fleets = 1, n_vessels = 1, n_species = 1, move_freq = 2)
```

`init_survey` *Initialise survey settings*

Description

`init_survey` is a function to mimic a fisheries-independent survey to sample catches from the populations.

Usage

```
init_survey(sim_init = NULL, design = "fixed_station", n_stations = 50,
  start_day = 90, stations_per_day = 5, Qs = NULL)
```

Arguments

`sim_init`	is the general simualtion settings from sim_init
`design`	is the survey design used, at the moment only *fixed_station*
`n_stations`	is a Numeric for the number of stations to be fished each. Note: If using 'fixed_station' design this will be rounded down to maintain a grid shape if not divisble.
`start_day`	is a Numeric for the first day of the survey each year

`stations_per_day`
: is a Numeric for the number of stations surveyed per day

`Qs`
: is a named Numeric Vector containing any survey catchabilities, assumed to be time invariant.

Value

is a list consisting of the survey setting and a a matrix for storing the log of catches from the survey, to be used as an input to run_sim.

Examples

```
init_survey(design = 'fixed_station', n_stations = 50, start_day = 90, stations_per_days = 5, Qs = c("spp1" = 0.1
```

`make_step` *make step function*

Description

`make_step` determines the new position of the vessel following a move, using the step distance and bearing inputs.

Usage

```
make_step(stepD, Bear, start.x, start.y)
```

Arguments

`stepD`	is a Numeric vector of the distance to move
`Bear`	is a Numeric vector of the bearing to move (in degrees)
`start.x`	is the starting point on the x-axis
`start.y`	is the starting point on the y-axis

Value

returns a new coordinate position through a vector (x, y)

Examples

```
make_step(stepD = 20, Bear = 90, start.x = 20, start.y = 5)
```

move_population *population movement function*

Description

move_population redistributes the population based on the movement probabilities

Usage

```
move_population(moveProp, StartPop)
```

Arguments

moveProp	is a list of the proportion of the population from each cell to reallocated to each of the other cells
StartPop	is a Numeric Matrix of the current populations distribution

Value

is a list of the new position for the population from each of the cells.

NOTE: This is not aggregated and requires calling the R function *Reduce('+', Lst)* to reaggregate. Would be better if done in function but Reduce is currently faster...but much more memory intensive to get out the lists...using the standard c++ accumulate function may work for this but untested

Examples

```
None at the moment
```

move_prob *movement probability function*

Description

move_prob calculates the movement probability between a cell and all other cells based on the distance and *lambda*.

Usage

```
move_prob(start, lambda, hab)
```

Arguments

start	is a Numeric vector of dim 2 for the starting position c(x,y)
lambda	is an integar for the value for the exponential decay in probability of movement, i.e. $Pr(B\|A) = \exp -\lambda * dist_{a,b} / Sum(c = 1 : c = n) \exp -\lambda * dist$
hab	is a matrix of the habitat suitability

Value

is a matrix of the movement probabilities from a cell

Examples

```
move_prob(c(2, 5), 0.3, matrix(nc = 3, runif(9)))
```

---

move_prob_Lst *movement probability function as a list*

Description

`move_prob_list` applies `move_prob` from all cells to all other cells and returns as a list.

Usage

```
move_prob_Lst(lambda, hab)
```

Arguments

lambda	is the decay value as in `move_prob`
hab	is a matrix of the habitat suitability for the population

Value

is a list of the movement probabilities form each cell to all other cells

Examples

```
None at the moment
```

---

norm_fun *Normal distribution*

Description

Helper function used for returning the PDF of a normal distribution from the supplied temperature tolerances in init_moveCov

Usage

```
norm_fun(x, mu, va)
```

Arguments

move_init is the output from init_moveCov

Examples

```
sapply(seq(2,20,0.1), mu = 10, va = 6)
```

---

plot_catch_comp *Plot the spatial catch composition from the commercial catches as 'square pie charts' using mapplots.*

Description

Plotting of spatial catch compositions at different levels of aggregation

Usage

```
plot_catch_comp(gran = c(20, 10, 5), logs = logs, fleets = 1:2,
  vessels = 1:5, trips = 1:20, years = 18:20, cluster_plot = FALSE,
  cluster_k = 5, scale_data = NULL)
```

Arguments

gran	is a Numeric Vector of granularities required
logs	is the fleet logs from combine_logs
fleets	is a Numeric Vector of the fleets to include in the catch composition plot
vessels	is a Numeric Vector of the vessels to include in the plot
trips	is a Numeric Vector of the trips to include
years	is a Numeric Vector of the years
cluster_plot	is a logical, determines whether also to run PAM cluserting on the catch compositions and plot the clusters spatially
scale_data	is a logical, whether to normalise the data before the clustering
clusters_k	is the number of clusters to search for in the PAM clustering algorithm

Examples

```
plot_catch_comp(gran = c(20,10,5,2), logs = logs, fleets = 1:2, vessels =
1:5. trips = 1:20, years = 18:20, cluster_plot = FALSE, cluster_k = 5,
scale_data = TRUE)
```

plot_daily_fdyn — *Plot daily fishing mortality dynamics*

Description

`plot_daily_fdyn` plots the daily fishing mortality dynamics by year.

Usage

```
plot_daily_fdyn(results)
```

Arguments

`results`	is output from the function run_sim.

Value

is a matplot of the daily fishing mortality dynamics

Examples

```
plot_daily_fdyn(results = results)
```

plot_fleet_trip — *Plot an entire fleet for a trip*

Description

`plot_fleet_trip` is a plot of a whole fleets vessels movement during one trip. It's intended for diagnostics.

Usage

```
plot_fleet_trip(logs = logs, fleet_no = 1, year_trip = 1, trip_no = 1)
```

Arguments

`logs`	is the combined log file, from combine_logs.
`fleet_no`	is a Numeric, the fleet from which to plot
`year_trip`	is a Numeric, the year in which the trip took place
`trip_no`	is a Numeric for the trip you wish to plot

Examples

```
plot_fleet_trip(logs = logs, fleet_no = 1, year_trip = 1, trip_no = 1)
```

plot_pop_summary *Plot population summary*

Description

plot_pop_summary plots the four population dynamic metrics: catches, biomass, fishing mortality and recruitment. It can either operate at a daily timestep or an annual timestep

Usage

```
plot_pop_summary(results = res, timestep = "daily", save = FALSE,
  save.location = ".")
```

Arguments

results	is an output from the function run_sim.
timestep	is a character string determining whether the plot is 'daily' or 'annual'
save	is a logical whether to save the plot
save.location	is a location (defaults to current directory)

Value

is a ggplot of all the species and metrics as a faceted plot examples plot_pop_summary(results = res, timestep = 'daily', save = TRUE, location = '.') ## Not run

plot_realised_stepF *Plot realised step function*

Description

plot_realised_stepF diagnostics plot of the step function shape realised in the simulation

Usage

```
plot_realised_stepF(logs = logs, fleet_no = 1, vessel_no = 1)
```

Arguments

logs	is the log file from combine_logs
fleet_no	is a Numeric of the fleet to plot
vessel_no	is a Numeric of the vessel to plot

Examples

```
plot_realised_stepF(logs = logs, fleet_no = 1, vessel_no = 1)
```

plot_spatiotemp_hab *Plot spatiotemporal habitat suitability*

Description

Function to plot out the habitat suitability, as adjusted by the spatiotemporal move covariates

Usage

```
plot_spatiotemp_hab(hab = NULL, moveCov = NULL, plot.file = getwd(),
  spwn_wk = NULL)
```

Arguments

hab	is the output from create_hab
moveCov	is the output from init_moveCov
plot.file	path to save the plots of the spatiotemporal habitats
spwn_wk	is a named list of the spawning week for each population

Examples

```
None
```

plot_survey *Plot the fisheries independent survey results*

Description

`plot_survey` plots the spatial abundances and an index from the fisheries independent survey, for each population.

Usage

```
plot_survey(survey = NULL, type = "spatial")
```

Arguments

survey	is the survey results from the run_sim function.
type	is a character indicating if *spatial* or *index*

Value

is a plot of the spatial distribution of survey catches and an inter-annual abundance index

Examples

```
plot_survey(survey = survey, type = "spatial")
```

plot_vessel_move *Plot vessel move*

Description

plot_vessel_move is a plot of a single vessel movement during one trip. It's intended for diagnostics.

Usage

```
plot_vessel_move(sim_init = NULL, logs = logs, fleet_no = 1,
  vessel_no = 1, year_trip = 1, trip_no = 1, fleets_init = NULL,
  pop_bios = NULL)
```

Arguments

logs	is the combined log file, from combine_logs.
fleet_no	is a Numeric, the fleet from which to plot
vessel_no	is a Numeric, the vessel to plot from the chosen fleet
year_trip	is a Numeric, the year in which the trip took place
trip_no	is a Numeric for the trip you wish to plot
fleets_init	is the output from init_fleet
pop_bios	is the output from run_sim when option save_pops_bios = TRUE

Examples

```
plot_vessel_move(sim_init = NULL, logs = logs, fleet_no = 1, vessel_no = 1, year_trip = 1,
trip_no = 1, fleets_init = NULL, pop_bios = NULL)
```

rad2deg *Radians to degrees*

Description

rad2deg is a helper function to covert radians to decimal degrees

Usage

```
rad2deg(r)
```

Arguments

d	is the bearing in decimal degrees

Value

is the bearing in radians

Examples

```
rad2deg(2)
```

Recr *Recruitment function*

Description

`Recr` returns a biomass of recruited fish to the population based on a stock-recruit relationship and some measure of variation.

Usage

```
Recr(model, params, B, cv, ..)
```

Arguments

model	is a character detailing the recruitment function to use (currently 'BH' for Beverton and Holt or 'Ricker' for a Ricker stock-recruit relationship.
params	is a Numeric vector of length 2, containing labelled *a* and *b* parameters for the stock-recruit function. For Beverton and Holt *a* refers to the maximum recruitment rate in biomass, *b* refers to the Spawning Stock Biomass (SSB) required to produce half the maximum. For Ricker *a* refers to the maximum productivity per spawner and *b* the density dependent reduction in productivity as the stock increases.
B	is a Numeric vector containing the SSB of the adult population from which the recruitment derives.
cv	is a Numeric vector containing the coefficient of variation in the recruitment function.

Value

returns the recruitment to the population in biomass.

Examples

```
Recr(model = 'BH', params = c("a" = 2000, "b" = 200), B = 1000, cv = 0.1)
```

Recr_mat *Recruitment function applied to matrix*

Description

Recr_mat returns a matrix of spatially referenced biomass of recruited fish to the population based on a stock-recruit relationship and some measure of variation.

Usage

```
Recr_mat(model, params, B, cv, ..)
```

Arguments

- model: is a character detailing the recruitment function to use (currently 'BH' for Beverton and Holt or 'Ricker' for a Ricker stock-recruit relationship.
- params: is a Numeric vector of length 2, containing labelled *a* and *b* parameters for the stock-recruit function. For Beverton and Holt *a* refers to the maximum recruitment rate in biomass, *b* refers to the Spawning Stock Biomass (SSB) required to produce half the maximum. For Ricker *a* refers to the maximum productivity per spawner and *b* the density dependent reduction in productivity as the stock increases.
- B: is a Numeric matrix containing the SSB of the adult population from which the recruitment derives.
- cv: is a Numeric vector containing the coefficient of variation in the recruitment function.

Value

returns the recruitment to the population in biomass.

Examples

```
Recr(model = 'BH', params = c("a" = 2000/4, "b" = 200/4), B =
matrix(c(1000,2000,500,750), nc = 2), cv = 0.1)
```

run_sim *Run sim*

Description

run_sim is the overarching simulation function, taking all the parameterised inputs and returning the results.

Usage

```
run_sim(sim_init = NULL, pop_init = NULL, move_cov = NULL,
  fleets_init = NULL, hab_init = NULL, InParallel = TRUE, cores = 1,
  save_pop_bio = FALSE, survey = NULL, closure = NULL, ...)
```

Arguments

sim_init	is the parameterised simulation settings from init_sim
pop_init	is the parameterised populations from init_pop
move_cov	is a parameterised movement covariate object, from init_moveCov
fleets_init	is the parameterised fleets from init_fleets
hab_init	is the parameterised habitat maps from create_hab
InParallel	is a BOLEEN indicating whether calculations should be done using parallel processing from parallel, default is TRUE
save_pop_bio	is a logical flag to indicate if you want to record #' true spatial population at each time step (day)
survey	is the survey settings from init_survey, else NULL if no survey is due to be simulated
closure	is the spatial closure settings from init_closurem else NULL if no closures are to be implemented

Value

is the results...

Examples

```
Not yet
```

---

step_length   *Step length function*

Description

step_length is a function to calculate the step length a vessel takes based on the step parameters provided for a gamma function and the revenue from the most recent fishing activity.

Usage

```
step_length(step_params = params[["step_params"]], revenue = revenue)
```

Arguments

step_params
: is a list of parameters which determine the relationship between revenue gained from the recent fishing activity and the next move step length, based on a gamma function. The list contains the following:
  - **rate** Determines the rate ....
  - **B1** Determines...
  - **B2** Determines ...
  - **B3** Determines ..

revenue
: is the last observed fishing revenue for the vessel

Value

step - the size of the next step

Examples

```
step_length(step_params = list(B1 = 1, B2 = 50, B3 = 2000, rate = 1),
revenue = 300)
```

---

sum_fleets_catches *Sum fleets catches*

Description

`sum_fleets_catches` is a helper function to apply `sum_fleet_catches` to all fleets, returning a single list of matrices with the catches of each population across all fleets and vessels.

Usage

```
sum_fleets_catches(FUN = sum_fleet_catches, fleets_log = NULL,
  sim_init = sim, ...)
```

Arguments

FUN
: is the function, i.e. `sum_fleet_catches`

fleets_log
: is the log of all the catches for all fleets, coming from application of go_fish_fleet to all fleets

n_spp
: is the number of populations in the simulation (NOTE: can remove this and take from the overall sim settings)

Value

is a list of matrices (one for each population) with all fleets catches of each population. This is then used as an input to the baranov calcs

Examples

```
spp_catches <- sum_fleets_catches(FUN = sum_fleet_catches,
fleets_log = applied_to_fleets, n_spp = 2)
```

---

sum_fleet_catches *Sum fleet catches*

Description

`sum_fleet_catches` is a helper function to take the spatial catches for an entire fleet and sum them as a matrix of catches for the fleet for each population

Usage

```
sum_fleet_catches(sim_init = sim, fleet_log = NULL)
```

Arguments

`sim_init`	is the initialised simulation settings, from `init_sim`
`fleet_log`	is the output of `go_fish_fleet`, i.e. the catch log information for a single fleet

Value

is a list of matrices (one for each population) with the entire fleets catches of the population

Examples

```
test <- sum_fleet_catches(fleet_log = applied_to_fleets[[1]])
```

---

test_step *test step length function*

Description

`test_step` is a function to test and review parameters for the step_length function. This is primarily to help with identifying the right parameters for the desired relationship between revenue and step length.

Usage

```
test_step(step_params = step_params, rev.max = 2000)
```

Arguments

`step_params` is a list of parameters which determine the relationship between revenue gained from the recent fishing activity and the next move step length, based on a gamma function. The list contains the following:

- **rate** Determines the rate ....
- **B1** Determines...
- **B2** Determines ...
- **B3** Determines ..

`rev.max` is the maximum revenue at which to test the step length function.

Value

is a plot of the relationship between revenue and step length

Examples

```
test_step(step_params = list(B1 = 1, B2 = 50, B3 = 2000, rate = 1), rev.max
= 2000)
```

Index