Package ‘metafolio’

February 20, 2015

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Metapopulation simulations for conserving salmon through portfolio optimization</td>
</tr>
<tr>
<td>Version</td>
<td>0.1.0</td>
</tr>
<tr>
<td>Date</td>
<td>2014-07-09</td>
</tr>
<tr>
<td>Description</td>
<td>The metafolio R package is a tool to simulate salmon metapopulations and apply financial portfolio optimization concepts. The package accompanies the paper 'Portfolio conservation of metapopulations under climate change'. See citation(``metafolio'`).</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Sean C. Anderson <a href="mailto:sean@seananderson.ca">sean@seananderson.ca</a></td>
</tr>
<tr>
<td>License</td>
<td>GPL-2</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://github.com/seananderson/metafolio">http://github.com/seananderson/metafolio</a></td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="http://github.com/seananderson/metafolio/issues">http://github.com/seananderson/metafolio/issues</a></td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.1.0)</td>
</tr>
<tr>
<td>LinkingTo</td>
<td>Rcpp, RcppArmadillo</td>
</tr>
<tr>
<td>Imports</td>
<td>Rcpp (&gt;= 0.11.2), plyr, colorspace, MASS</td>
</tr>
<tr>
<td>Suggests</td>
<td>knitr, devtools, TeachingDemos, RColorBrewer, reshape2</td>
</tr>
<tr>
<td>Author</td>
<td>Sean C. Anderson [aut, cre], Jonathan W. Moore [ctb], Michelle M. McClure [ctb], Nicholas K. Dulvy [ctb], Andrew B. Cooper [ctb]</td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>yes</td>
</tr>
<tr>
<td>Repository</td>
<td>CRAN</td>
</tr>
<tr>
<td>Date/Publication</td>
<td>2014-07-10 08:45:42</td>
</tr>
</tbody>
</table>
R topics documented:

add_dens_polygon .................................................. 3
annotate ................................................................. 3
count_quasi_exts .................................................... 4
create_asset_weights ................................................. 5
custom_bw ............................................................... 6
CVaR ................................................................. 6
est_beta_params ....................................................... 6
fastlm ................................................................. 7
fit_ricker .............................................................. 7
generate_env_ts ...................................................... 8
generate_straying_matrix .......................................... 9
get_conserv_plans_mv ............................................... 9
get_efficient_frontier .............................................. 10
get_port_vals .......................................................... 11
get_quanlile_contour ............................................... 11
gg_color_hue .......................................................... 12
impl_error ............................................................ 12
is_element ............................................................. 13
is_quasi_ext ........................................................... 14
metafolio ............................................................... 14
metasim_base ......................................................... 15
meta_sim ............................................................... 16
monte_carlo_portfolios ............................................. 19
my.axis ............................................................... 20
optim_thermal ........................................................ 20
plot_cons_plans ...................................................... 21
plot_correlation_between_returns ................................ 22
plot_efficient_portfolios ......................................... 22
plot_panel_lines ................................................... 24
plot_rickers .......................................................... 24
plot_sim_ts ........................................................... 25
plot_sp_A_ts ........................................................ 26
ricker ................................................................. 27
ricker_escapement .................................................. 28
ricker_v_t ............................................................. 28
run_cons_plans ...................................................... 29
thermal_area .......................................................... 31
thermal_curve_a ..................................................... 31
thermal_integration ............................................... 32
VaR ................................................................. 33

Index 34
add_dens_polygon

Description
Add a kernel density polygon

Usage
add_dens_polygon(x, y, col, lwd = 1.7, alpha = c(0.25, 0.75),
    add pts = FALSE, add_poly = TRUE)

Arguments
- x: x values
- y: y values
- col: Colour to add polygon with. Will be made into two levels of opacity.
- lwd: lwd Line width
- alpha: A numeric vector of length 2 that gives the confidence levels for the two kernel
density polygons.
- add_poly: Add polygons?
- add pts: Logical: should points be added?

annotate

Description
Add annotations to panel

Usage
annotate(label, xfrac = 0.008, yfrac = 0.18, pos = 4, cex = 0.9, ...)

Arguments
- label: The text to add as a label
- xfrac: Fraction over from the left
- yfrac: Fraction down from the top
- pos: Position of text to pass to text
- cex: Character expansion value to pass to text
- ...: Anything else to pass to text
count_quasi_exts

Take \texttt{meta_sim} output objects and count quasi extinctions

\textbf{Description}

Take \texttt{meta_sim} output objects and count quasi extinctions

\textbf{Usage}

\begin{verbatim}
count_quasi_exts(dat, quasi_thresh, ignore_pops_thresh = 5, duration = 1)
\end{verbatim}

\textbf{Arguments}

\begin{description}
\item[dat] Input data. Should be a list of lists. The first level corresponds to the conservation plan and the second level corresponds to the replicate.
\item[quasi_thresh] The quasi extinction threshold
\item[ignore_pops_thresh] Threshold below which to ignore populations (e.g. if you started some populations with very low abundance and you don’t want to count those populations.
\item[duration] Number of years that the abundance must be below the quasi\_thresh before being counted as quasi extinct.
\end{description}

\textbf{Value}

A list of matrices. The list elements correspond to the conservation plans. The columns of the matrix correspond to the subpopulations that were above the ignore\_pops\_thresh level. The rows of the matrix correspond to the replicates.

\textbf{Examples}

\begin{verbatim}
## Not run:
set.seed(1)
w_plans <- list()
w_plans[[1]] <- c(5, 1000, 5, 1000, 5, 5, 1000, 5, 1000, 5)
w_plans[[2]] <- c(5, 5, 1000, 1000, 1000, 1000, 1000)
w_plans[[3]] <- c(rep(1000, 4), rep(5, 6))
w_plans[[4]] <- rev(w_plans[[3]])
plans_name_sp <- c("Full range of responses", "Most stable only", "Lower half", "Upper half")
n_trials <- 50 # number of trials at each n conservation plan n_plans <- 4 # number of plans num_pops <- c(2, 4, 8, 16) # n pops to conserve w <- list()
for(i in 1:n_plans) { # loop over number conserved w[[i]] <- list()
    for(j in 1:n_trials) { # loop over trials
        w[[i]][[j]] <- matrix(rep(625, 16), nrow = 1)
        w[[i]][[j]][-sample(1:16, num_pops[i])] <- 5
    }
}
\end{verbatim}
create_asset_weights

Create an asset weights matrix

Description

Create an asset weight matrix to run through the Monte Carlo algorithm and test possible portfolios.

Usage

create_asset_weights(n_pop, n_sims, weight_lower_limit = 0.02)

Arguments

n_pop The number of subpopulations.
n_sims The number of simulations.
weight_lower_limit The lowest fraction allowed for a subpopulation weight. For example, a value of 0.02 means a subpopulation will at least be assigned 2% of the total capacity

Value

A matrix. The columns represent subpopulations. The rows represent simulation repetitions.

Examples

create_asset_weights(n_pop = 5, n_sims = 10, weight_lower_limit = 0.001)
custom_bw  
*Custom bandwidth*

**Description**
Based on bandwidth.nrd from **MASS**. This version takes the absolute value of `var` to avoid errors.

**Usage**
```
custom_bw(x)
```

**Arguments**
- `x` A numeric vector

---

cVaR  
*Conditional Value at Risk*

**Description**
Get the conditional value at risk.

**Usage**
```
cVaR(x, probs = 0.05)
```

**Arguments**
- `x` A numeric vector
- `probs` The probability cutoff to pass to the CVaR function.

---

est_beta_params  
*Get beta parameters from mean and variance*

**Description**
Get beta parameters from mean and variance

**Usage**
```
est_beta_params(mu, var)
```

**Arguments**
- `mu` Mean
- `var` Variance
**fastlm**

Super fast linear regression

**Description**

Super fast linear regression

**Usage**

`fastlm(yr, Xr)`

**Arguments**

- `yr`: Vector of y values
- `Xr`: Model matrix

**fit_ricker**

Fit Ricker linear regression

**Description**

Fit a Ricker curve to spawner-recruit data and return the intercept (a) and slope (b). The model is fit via the `RcppArmadillo` package for speed.

**Usage**

`fit_ricker(S, R)`

**Arguments**

- `S`: Spawners as a numeric vector.
- `R`: Recruits or returns as a numeric vector.

**Value**

A named list with components `a` for the intercept and `b` for the slope.

**Examples**

```r
S <- seq(100, 1000, length.out = 100)
v_t <- rnorm(100, 0, 0.1)
R <- mapply(ricker_v_t, spawners = S, v_t = v_t, a = 1.9, b = 900, d = 1)
plot(S, log(R/S))
fit_ricker(S, R)
```
generate_env_ts

Create an environmental time series.

Description

Generate various types of environmental time series.

Usage

generate_env_ts(n_t, type = c("sine", "arma", "regime", "linear", "linear_arma", "constant"),
sine_params = list(amplitude = 1, ang_frequency = 0.2, phase = 0, mean_value = 0, slope = 0, sigma_env = 0.02),
arma_params = list(mean_value = 0, sigma_env = 0.5, ar = 0.4, ma = 0),
regime_params = list(break_pts = c(25, 75), break_vals = c(-1, 0, 1)),
linear_params = list(min_value = -1, max_value = 1, sigma_env = 0.1, start_t = 1),
linear_arma_params = list(min_value = -1, max_value = 1, sigma_env = 0.1, start_t = 1, ar = 0.4, ma = 0),
constant_params = list(value = 0))

Arguments

n_t Length of time series.
type Type of time series to produce.
sine_params Parameters controlling sine wave time series.
arma_params Parameters controlling ARMA time series.
regime_params Parameters controlling regime-shift time series.
linear_params Parameters controlling warming or cooling time series. Minimum environmental value, maximum environmental value, environmental standard deviation, and the year to start the linear trend (useful if you're going to throw out the early years as burn in).
linear_arma_params A combination of arma_params and linear_params.
constant_params Parameter controlling constant time series.

Examples

types <- c("sine", "arma", "regime", "linear", "linear_arma", "constant")
x <- list()
for(i in 1:6) x[[i]] <- generate_env_ts(n_t = 100, type = types[i])
op <- par(mfrow = c(5, 1), mar = c(3,3,1,0), cex = 0.7)
for(i in 1:6) plot(x[[i]], type = "o", main = types[i])
par(op)
generate_straying_matrix

Generate a matrix of straying proportions within a metapopulation

Description

Generate a matrix of straying proportions within a metapopulation. Based on Eq. 2 in Cooper and Mangel (1999).

Usage

generate_straying_matrix(n_pop, stray_fraction, stray_decay_rate)

Arguments

n_pop Number of subpopulations.
stray_fraction Fraction of individuals that stray from a given subpopulation.
stray_decay_rate Exponential rate that straying decays with distance between subpopulations.

References


Examples

x <- generate_straying_matrix(10, 0.01, 0.3)
image(x, col = rev(heat.colors(12)))

generate_straying_matrix

get_conserv_plans_mv

Run simulation for conservation schemes

Description

Run the metapopulation simulation for various conservation prioritization schemes.

Usage

get_conserv_plans_mv(weights, reps = 150, assess_freq = 5, burn = 1:30, risk_fn = var, ...)

get_conserv_plans_mv

Run simulation for conservation schemes

Description

Run the metapopulation simulation for various conservation prioritization schemes.

Usage

get_conserv_plans_mv(weights, reps = 150, assess_freq = 5, burn = 1:30, risk_fn = var, ...)

get_conserv_plans_mv
get_efficient_frontier

Get the efficient frontier from mean and variance values

Arguments

weights A matrix of habitat weights. Each row corresponds to another scenario. Each column is a different habitat location.

reps Number of portfolios to simulate.

assess_freq The frequency (in generations) of spawner-recruit re-assessment. Passed to meta_sim.

burn Cycles to throw out as burn in.

risk_fn Type of variance or risk metric. By default takes the variance. Instead you can supply any function that takes a numeric vector and returns some single numeric value. E.g. CVaR.

... Other values to pass to meta_sim.

Value

Returns the portfolio mean and variance values and the simulation runs.

get_efficient_frontier

get_efficient_frontier(m, v)

Arguments

m A vector of mean values

v A vector of variance values
get_port_vals

Get portfolio mean and variance values

Description

Takes a list created by `meta_sim` and returns the mean and variance (or risk metric) values. This function is used by other internal functions, but can also be used as its own low-level function.

Usage

```r
get_port_vals(x, risk_fn = var, burn = 1:30)
```

Arguments

- `x`: A list object as returned from `meta_sim`
- `burn`: Number of years to throw out as burn in
- `risk_fn`: Type of variance or risk metric. By default takes the variance. Instead you can supply any function that takes a numeric vector and returns some single numeric value. E.g. CVaR.

Value

A data frame with columns for the mean (m) and variance (v).

See Also

- `get_conserv_plans_mv`, `plot_cons_plans`

Examples

```r
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
base1 <- meta_sim(n_pop = 10, env_params = arma_env_params, env_type = "arma", assess_freq = 5)
get_port_vals(base1)
```

get_quantile_contour

Get quantile contour

Description

Get quantile contour

Usage

```r
get_quantile_contour(x, alpha = 0.8)
```
Arguments

- x: Output from kde2d.
- alpha: The quantile level.

---

**gg_color_hue**

**ggplot2-like colour scale in HCL space**

Description

ggplot2-like colour scale in HCL space

Usage

```r
gg_color_hue(n, hue_min = 10, hue_max = 280, l = 62, c = 100)
```

Arguments

- `n`: Number of colours to return.
- `hue_min`: Minimum hue value in the range [0,360]
- `hue_max`: Maximum hue value in the range [0,360]
- `l`: Luminance in the range [0,100]
- `c`: Chroma of the colour.

Details

See the `hcl` function for details.

Examples

```r
gg_color_hue(10)
```

---

**impl_error**

*Add implementation error*

Description

Add implementation error with a beta distribution.

Usage

```r
impl_error(mu, sigma_impl)
```
is_element

Arguments

mu
The mean
sigma_impl
Implementation error standard deviation

Value

A single numeric values representing a sample from a beta distribution with the specified mean and standard deviation.

References


Examples

y <- sapply(1:200, function(x) impl_error(0.5, 0.2))
hist(y)

y <- sapply(1:200, function(x) impl_error(0.3, 0.1))
hist(y)

is_element Check if x is an element of y.

Description

Check if x is an element of y.

Usage

is_element(x, y)

Arguments

x
An integer to check
y
A vector to check if x is an element of y.
is_quasi_ext  

Return whether there was an instance of quasi extinction

Description

Return whether there was an instance of quasi extinction

Usage

is_quasi_ext(x, thresh, duration = 1)

Arguments

x  
A numeric vector.

thresh  
The quasi-extinction threshold in absolute numbers.

duration  
The number of years below the threshold before a quasi extinction has occurred.

Examples

x <- seq(100, 0, length.out = 20)
is_quasi_ext(x, thresh = 10)

Description

The metafolio R package is a tool to simulate metapopulations and apply financial portfolio optimization concepts. The package was originally written for salmon simulations, so some of the language refers to salmon-specific terminology, but the package could be used and/or adopted for other taxonomic groups.

Details

The main simulation function is meta_sim. This function takes care of running an individual simulation iteration. The package also contains functions for exploring conservation scenarios with these simulations (see the "Assessing multiple conservation scenarios" section below), and find optimal conservation strategies (see the "Portfolio optimization section" below).

Running a simulation once

To run a single simulation iteration, see the function meta_sim. To plot the output from one of these simulations, see the function plot_sim_ts.
Assessing multiple conservation scenarios

You can use `run_cons_plans` to run `meta_sim` for multiple iterations and across multiple conservation strategies. These strategies could focus on the spatial distribution of conservation or on the number of populations conserved.

The function `plot_cons_plans` can plot the output from `run_cons_plans`.

Specifying environmental patterns

When you run `meta_sim` you can specify the environmental signal. One of the arguments is a list of options to pass to `generate_env_ts`, which controls the environmental pattern.

Diagnostic plots

`metafolio` contains some additional plotting functions to inspect the spawner-return relationships and the correlation between returns: `plot_rickers`, and `plot_correlation_between_returns`.

Portfolio optimization

`metafolio` also contains some experimental functions for finding optimal conservation strategies (an efficient frontier). This is analogous to financial portfolio where the goal is to find the investment weights that maximizes expected return for a level of expected risk, or vice-versa. Presently, these functions rely on Monte Carlo sampling, and so are rather slow.

For this purpose, the function `create_asset_weights` can generate a matrix of asset weights, which can then be passed to `monte_carlo_portfolios` to do the optimization itself. `plot_efficient_portfolios` can be used to plot the optimization output.

See the package vignette vignette("metafolio") for more extensive explanation of how to use `metafolio` along with some examples.

---

**metasim_base**

*Base-level metapopulation simulation function*

---

**Description**

This is an Rcpp implementation of the main simulation. It is meant to be called by `meta_sim`.

**Usage**

```r
metasim_base(n_pop, n_t, spawners_0, b, epsilon_mat, A_params, add_straying, stray_mat, assess_years, r_escp_goals, sigma_Impl, add_Impl_error, decrease_b, debug)
```
Arguments

- **n_t**: The number of years.
- **n_pop**: Number of populations.
- **spawners_0**: A vector of spawner abundances at the start of the simulation. Length of the vector should equal the number of populations.
- **b**: Ricker density-dependent parameter. A vector with one numeric value per population.
- **epsilon_mat**: A matrix of recruitment deviations.
- **A_params**: A matrix of Ricker a parameters.
- **add_straying**: Implement straying between populations?
- **stray_mat**: A straying matrix.
- **assess_years**: A vector of years to assess a and b in.
- **r_escp_goals**: A matrix of escapement goals.
- **sigma_impl**: Implementation standard deviation for the implementation error beta distribution.
- **add_impl_error**: Add implementation error? Implementation error is derived using `impl_error`.
- **decrease_b**: A numeric value to decrease all streams by each generation. This is intended to be used to simulate habitat loss, for example though stream flow reduction with climate change.
- **debug**: Boolean. Should some debugging messages be turned on?

**meta_sim**  
*Run a single metapopulation simulation.*

Description

This is the master function for running `metafolio` simulations. It runs a single iteration of a simulation. The arguments can be manipulated with other functions in the package to use this function as part of a portfolio analysis.

Usage

```r
meta_sim(n_t = 130, n_pop = 10, stray_decay_rate = 0.1,
stray_fraction = 0.02, b = rep(1000, n_pop), spawners_0 = round(b),
sigma_v = 0.7, v_rho = 0.4, a_width_param = c(seq(0.08, 0.04, length.out = n_pop/2), rev(seq(0.08, 0.04, length.out = n_pop/2))),
optim_temp = seq(13, 19, length.out = n_pop),
max_a = thermal_integration(n_pop), env_type = c("sine", "arma", "regime", "linear", "constant"), env_params = list(amplitude = 3.2, ang_frequency = 0.2, phase = runif(1, -pi, pi), mean_value = 15, slope = 0, sigma_env = 0.3),
start_assessment = 20, a_lim = c(0.02, 4), b_lim = c(0.5, 1.5),
silence_warnings = TRUE, sigma_impl = 0.1, assess_freq = 10,
use_cache = FALSE, cache_env = FALSE, add_straying = TRUE,
add_impl_error = TRUE, skip_saving_cache = FALSE, decrease_b = 0,
debug = FALSE)
```
Arguments

\begin{itemize}
    \item \texttt{n_t} The number of years.
    \item \texttt{n_pop} Number of populations
    \item \texttt{stray\_decay\_rate} Rate that straying (exponentially) decays with distance.
    \item \texttt{stray\_fraction} Fraction of fish that stray from natal streams.
    \item \texttt{b} Ricker density-dependent parameter. A vector with one numeric value per population.
    \item \texttt{spawners\_0} A vector of spawner abundances at the start of the simulation. Length of the vector should equal the number of populations.
    \item \texttt{sigma\_v} Stock-recruit residual standard deviation of the log-deviations.
    \item \texttt{v\_rho} AR1 serial correlation of stock-recruit residuals.
    \item \texttt{a\_width\_param} Width of the thermal curves by population.
    \item \texttt{optim\_temp} Optimal temperatures by population.
    \item \texttt{max\_a} Maximum Ricker productivity parameters (a) by population. The value obtained at the optimum temperature. Note how the default argument uses the \texttt{thermal\_integration} function.
    \item \texttt{env\_type} The type of environmental time series to generate. One of "sine", "arma", "regime", "linear", or "constant". See \texttt{generate\_env\_ts}.
    \item \texttt{env\_params} Parameters to pass on to \texttt{generate\_env\_ts}. You must provide the appropriate list given your chosen type of environmental signal.
    \item \texttt{start\_assessment} Generation to start estimating the stock recruit relationship for escapement targets. The assessment is carried out using \texttt{fit\_ricker}.
    \item \texttt{a\_lim} A vector of length two giving the lower and upper limits for Ricker a values. If a value is estimated beyond these limits it will be set to the limit value.
    \item \texttt{b\_lim} A vector of length two giving the lower and upper limits for the estimated Ricker b values *as fractions* of the previously assessed value. If a value is estimated beyond these limits it will be set to the limit value.
    \item \texttt{silence\_warnings} Should the warnings be skipped if the Ricker a or b values exceed their specified bounds? \texttt{meta\_sim} will still print other warnings regardless of this argument value.
    \item \texttt{sigma\_impl} Implementation standard deviation for the implementation error beta distribution.
    \item \texttt{assess\_freq} How many generations before re-assessing Ricker a and b parameters.
    \item \texttt{use\_cache} Use the stochastically generated values (stock-recruit residuals and possibly environmental time series) from the previous run? See the Details section below.
    \item \texttt{cache\_env} Logical: Should the environmental time series be cached? If \texttt{use\_cache = TRUE} then this will automatically happen. But, you could set \texttt{cache\_env = TRUE} and \texttt{use\_cache = FALSE} to only cache the environmental time series. See the Details section below.
\end{itemize}
add_straying  Implement straying between populations?
add_impl_error Add implementation error? Implementation error is derived using `impl_error`.
decrease_b A numeric value to decrease all streams by each generation. This is intended to
be used to simulate habitat loss, for example though stream flow reduction with
climate change.
skip_saving_cache Logical: if TRUE then no data will be cached for the next iteration. This will save
time when running many simulations.
debug Logical: if TRUE then `meta_sim` will print a number of debugging statements
while it runs.

Details
To use either of the caching options, you must have run `meta_sim` at least once in the current
session with both caching arguments set to FALSE to generate the cached values first. If you’re
running many iterations of `meta_sim` and you want to cache, then the first iteration should have
both cache arguments set to FALSE, and subsequent runs can set one or both to TRUE. Internally,
`meta_sim` caches by writing the appropriate data to an `.rda` file in a temporary directory.

Value
A list is returned that contains the following elements. All matrices that are returned (except the
straying matrix) feature populations along the columns and generations/years along the rows.

A  A matrix of abundances.
F  A matrix of fishing mortality in numbers.
E  A matrix of realized escapement.
Eps A matrix of (log) spawner-return residuals. These have been log-normal bias corrected so their
expected value after exponentiation will be one.
A_params  A matrix of actual Ricker a parameters.
Strays_leaving  A matrix of strays leaving.
Strays_joining  A matrix of strays joining.
ev_ts A vector of the environmental time series.
stray_mat The straying matrix. These fractions are constant across generations/years. Rows and
columns are populations.
n_pop The total possible populations as input in the simulation.
n_t The number of generations/years the simulation was run for.
b The original Ricker b values as specified.
est_a A matrix of estimated Ricker a values.
est_b A matrix of estimated Ricker b values.
**monte_carlo_portfolios**

**Examples**

```r
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
base1 <- meta_sim(n_pop = 10, env_params = arma_env_params,
  env_type = "arma", assess_freq = 5)

plot_sim_ts(base1, years_to_show = 70, burn = 30)
```

---

**monte_carlo_portfolios**

*Monte Carlo asset weights into portfolios*

**Description**

Monte Carlo the asset weights into portfolios and record the simulation output and portfolio metrics (mean and variance).

**Usage**

```r
monte_carlo_portfolios(weigths_matrix, n_sims = 500, mean_b = 1000,
  burn = 1:30, ...)
```

**Arguments**

- `weights_matrix`: A matrix of asset weights. The columns correspond to the different assets and the rows correspond to the simulation iterations.
- `n_sims`: The number of simulations to run.
- `mean_b`: The mean Ricker capacity value.
- `burn`: The number of years to discard as burn in.
- `...`: Anything else to pass to `meta_sim`.

**Value**

A list object with three elements: 
- `port_vals` (a matrix with a column of mean rate of change and variance of rate of change),
- `n_sims` (the number of simulations ran), and
- `sims_out` (a list in which each element corresponds to the output from the run of `meta_sim`).

**See Also**

`meta_sim`, `create_asset_weights`

**Examples**

```r
weights_matrix <- create_asset_weights(n_pop = 4, n_sims = 3,
  weight_lower_limit = 0.001)
mc_ports <- monte_carlo_portfolios(weights_matrix = weights_matrix,
  n_sims = 3, mean_b = 1000)
```
### my.axis

*Add a pretty axis*

**Description**

Add a pretty axis

**Usage**

```r
my.axis(side, shade_years = NULL, ylab = "", yticks = NA)
```

**Arguments**

- `side`: Number indicating the side to add an axis (as in the side number passed to `axis`).
- `shade_years`: An optional numerical vector of length two giving the minimum and maximum years over which to add a light grey shading.
- `ylab`: Y axis label
- `yticks`: Logical: should y-axis ticks be added?

### optim_thermal

*Optimize to find optimal max productivity Ricker a*

**Description**

Optimize to find optimal max productivity Ricker a

**Usage**

```r
optim_thermal(optim_temp, width_param, desired_area)
```

**Arguments**

- `optim_temp`: The optimum temperature as a numeric value
- `width_param`: The width parameter as a numeric value
- `desired_area`: The desired area as a numeric value
plot_cons_plans

Plot conservation plans in mean-variance space

Description

This makes a mean-variance plot of the portfolio output. It can take care of: plotting the individual portfolios, adding 2D kernel density polygons at two quantile levels, and adding an efficient frontier.

Usage

plot_cons_plans(plans_mv, plans_name, cols, xlim = NULL, ylim = NULL, add_pts = TRUE, add_all_efs = FALSE, x_axis = TRUE, y_axis = TRUE, add_legend = TRUE, legend_pos = "topright", w_show = "all", xlab = "Variance", ylab = "Mean", add_poly = TRUE, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plans_mv</td>
<td>The plans_mv element of the output from run_cons_plans.</td>
</tr>
<tr>
<td>plans_name</td>
<td>A character vector of what to label each conservation plan.</td>
</tr>
<tr>
<td>cols</td>
<td>Colours for the conservation plan polygons.</td>
</tr>
<tr>
<td>xlim</td>
<td>X limits</td>
</tr>
<tr>
<td>ylim</td>
<td>Y limits</td>
</tr>
<tr>
<td>add_pts</td>
<td>Logical: add the points?</td>
</tr>
<tr>
<td>add_all_efs</td>
<td>Logical: add efficient frontiers?</td>
</tr>
<tr>
<td>x_axis</td>
<td>Logical: add x axis?</td>
</tr>
<tr>
<td>y_axis</td>
<td>Logical: add y axis?</td>
</tr>
<tr>
<td>add_legend</td>
<td>Logical: add y legend?</td>
</tr>
<tr>
<td>legend_pos</td>
<td>A character string to pass to legend denoting the position of the legend.</td>
</tr>
<tr>
<td>w_show</td>
<td>If &quot;all&quot; then all plans will be shown. If a numeric vector, then those plans will be shown. E.g. c(1, 3) will only show the first and third plans.</td>
</tr>
<tr>
<td>xlab</td>
<td>X axis label.</td>
</tr>
<tr>
<td>ylab</td>
<td>Y axis label.</td>
</tr>
<tr>
<td>...</td>
<td>Anything else to pass to plot.default.</td>
</tr>
<tr>
<td>add_poly</td>
<td>Add the kernal smoother quantile polygons?</td>
</tr>
</tbody>
</table>

Value

A plot. Also, the x and y limits are returned invisibly as a list. This makes it easy to make the first plot and then save those x and y limits to fix them in subsequent (multipanel) plots.
plot_correlation_between_returns

Plot correlation of returns (i.e. metapopulation abundance) across stocks.

Description

Create a matrix plot showing the correlation between the log returns of each stock/asset.

Usage

plot_correlation_between_returns(x, burn = 1:30,
    pal = rev(gg_color_hue(x$n_pop)),
    xlab = "log of return abundance by population",
    ylab = "log of return abundance by population")

Arguments

- x: A list output object from `meta_sim`.
- burn: Number of years to discard at start as burn in.
- pal: Colours to label each stock/asset.
- xlab: X axis label
- ylab: Y axis label

Examples

arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
basel <- meta_sim(n_pop = 10, env_params = arma_env_params, env_type = "arma", assess_freq = 5)
plot_correlation_between_returns(basel)

plot_efficient_portfolios

Basic plot of efficient portfolio and asset contributions

Description

This function creates a mean-variance plot of the portfolios across possible asset weights, colour the efficient frontier, and show the contribution of the different stocks/assets. It also (invisibly) returns the values that make up the plot so you can create your own custom plots with the data. See the Returns section for more details.
plot_efficient_portfolios

Usage

plot_efficient_portfolios(port_vals, weights_matrix, pal, plot = TRUE,
                           ylab_dots = "Mean of metapopulation growth rate",
                           xlab_dots = "Variance of metapopulation growth rate",
                           ylab_bars = "Percentage", xlab_bars = "Variance (multiplied by 1000)",
                           port_cols = c("grey50", "red"), pch = 19, ...)

Arguments

port_vals A matrix of means and variances (down the two columns). This likely comes
from the output of monte_carlo_portfolios.
weights_matrix The same weight matrix that was passed to monte_carlo_portfolios.
pal Colour palette for the stocks/assets in the barplot.
plot Logical: should the plots be made?
ylab_dots Y axis label for the mean-variance scatterplot.
xlab_dots X axis label for the mean-variance scatterplot.
ylab_bars Y axis label for the barplot.
xlab_bars X axis label for the barplot.
port_cols Colours for the dots. A vector of colours for the non-efficient and efficient port-
folios.
pch Dot type
... Anything else to pass to both plot.default and barplot.

Value

A two panel plot and an (invisible) list of values calculated within the function. This list contains
pv (mean, variance, and whether it was part of the efficient frontier); ef_port_ids (the portfolio
IDs [run numbers] that are part of the efficient frontier; min_var_port_id (the portfolio ID for the
minimum-variance portfolio); ef_weights (the weights of the portfolios on the efficient frontier).

Examples

## Not run:
weights_matrix <- create_asset_weights(n_pop = 6, n_sims = 3000,
                                       weight_lower_limit = 0.001)
mc_ports <- monte_carlo_portfolios(weights_matrix = weights_matrix,
                                     n_sims = 3000, mean_b = 1000)

col_pal <- rev(gg_color_hue(6))
ef_dat <- plot_efficient_portfolios(port_vals = mc_ports$port_vals,
                                     pal = col_pal, weights_matrix = weights_matrix)

names(ef_dat)

## End(Not run)
plot_panel_lines

Description

Standard matrix plot of values by stream for one panel:

Usage

plot_panel_lines(dat, ymin = c("zero", "min"), ystretch = 1.1, ...)

Arguments

dat The matrix of values to plot
ymin Minimum y value for axis
ystretch A fraction to multiply the max value of when setting the y axis limits. This is useful to make space for a panel label within the plot.
... Anything else to pass to matplot.

plot_rickers

Plot sample Ricker curves for each stock

Description

Make a plot of Ricker curves for each stock. Can be useful for visualizing how the simulation parameters are impacting the Ricker curves and how these vary with temperature across stocks. The colour of the lines corresponds to the relative thermal tolerance of that stock. The shaded region shows the range of spawners observed throughout the simulations.

Usage

plot_rickers(x, pal = rep("black", x$n_pop), n_samples = 40,
    add_y_axes_pops = c(1, 6), add_x_axes_pops = c(6:10), burn = 1:30,
    add_shading = TRUE, ...)

Arguments

x Output list from meta_sim.
pal Colours for stocks.
n_samples Number of sample lines to draw from the a parameters.
add_y_axes_pops Panels to add y axes on.
add_x_axes_pops Panels to add x axes on.
plot_sim_ts

- **burn**: Number of initial years to throw out as burn in.
- **add_shading**: Logical: add the light grey shading for the range of observed spawner abundance?
- ... Anything else to pass to `plot.default`.

**Examples**

```r
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
base1 <- meta_sim(n_pop = 10, env_params = arma_env_params, env_type = "arma", assess_freq = 5)
plot_rickers(base1)
```

---

**plot_sim_ts**  
Plot various time series from a simulation run

**Description**

This function lets you quickly visualize the time series of output from a simulation run.

**Usage**

```r
plot_sim_ts(x, pal = rev(gg_color_hue(x$n_pop)), years_to_show = 30,
            burn = 1:50, shade_years = NULL, adj = 0.02, add_units = FALSE,
            yticks = rep(list(NA), 10), oma = c(4, 4.5, 1, 1))
```

**Arguments**

- **x**: A list output object from a simulation run of `meta_sim`.
- **pal**: A colour palette for the lines. One colour per line (each line is a population time series).
- **years_to_show**: How many years to plot after the burn in period.
- **burn**: The number of years to discard as burn in at the beginning of the time series.
- **adj**: Adj parameter to pass to `mtext` for panel labels.
- **shade_years**: Shade some years? Give a vector. Shading will be applied from the minimum to maximum value. Can be used to show burn in period.
- **add_units**: Should the units be added to the y axis?
- **yticks**: Position of ticks on the Y axis.
- **oma**: oma vector to pass to `par` for outer margin space.

**Examples**

```r
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
base1 <- meta_sim(n_pop = 10, env_params = arma_env_params, env_type = "arma", assess_freq = 5, decrease_b = 10)
plot_sim_ts(base1, years_to_show = 70, burn = 1:30)
```
plot_sp_A_ts

Plot sample time series from a portfolio simulation

Description

Plot sample time series from a portfolio simulation

Usage

plot_sp_A_ts(xL ylimL x_axis [] trueL y_axis [] trueL rate [] falseL
lwd [] 1.7L y_axis_ticks [] nullL start_new_plots [] 1L labels [] nullL
burn [] SPL add_lm [] falseL cols [] NNNI

Arguments

x Object to plot. Should be a list of outputs from meta_sim.
ylim Y axis limits.
x_axis Should an x axis be added?
y_axis Should a y axis be added?
rate If TRUE then the first difference (rate of change) will be plotted. If FALSE then
the raw data will be plotted.
lwd Line width of the lines.
y_axis_ticks Location of the y-axis tick marks, if you want to specify them.
start_new_plots On which elements of the list x should new panels be started? A numeric vector.
labels Labels for the panels.
burn Burn in period to discard.
add_lm Add a regression trend line?
cols Colours for the lines. A vector of character.
... Anything else to pass to plot.default

Value

A plot, possibly with multiple panels.

Examples

w_plans <- list()
w_plans[[1]] <- c(5, 1000, 5, 1000, 5, 5, 1000, 5, 1000, 5)
w_plans[[2]] <- c(5, 5, 1000, 1000, 1000, 1000, 5, 5, 5)
w_plans[[3]] <- c(rep(1000, 4), rep(5, 6))
w_plans[[4]] <- rev(w_plans[[3]])
w <- list() for(i in 1:4) { # loop over plans


```r
w[[i]] <- list()
for(j in 1:2) { # loop over trials
  w[[i]][[j]] <- matrix(w_plans[[i]], nrow = 1)
}
}

cons_arma_ts <- list()
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)
for(i in 1:4) {
  use_cache <- ifelse(i == 1, FALSE, TRUE)
  cons_arma_ts[[i]] <- meta_sim(b = w[[i]][[1]], n_pop = 10, env_params = arma_env_params, env_type = "arma", assess_freq = 5,
      use_cache = use_cache)
}
cols <- RColorBrewer::brewer.pal(5, "Dark2")
par(mfrow = c(1, 1))
plot_sp_a_ts(cons_arma_ts, ylim = c(0000, 12400),
    start_new_plots = c(1, 3),
    labels = c("Balanced response diversity", "ignore", "Unbalanced response diversity", "ignore"), cols = cols)
```

---

**ricker**

*A simple Ricker model*

---

**Description**

A simple Ricker model

**Usage**

```r
ricker(spawners, a, b)
```

**Arguments**

- `spawners` : Spawner abundance
- `a` : Ricker productivity parameter. Recruits are e^a at the origin.
- `b` : Ricker density dependent parameter.

**Value**

Returns the number of recruits.

**Examples**

```r
S <- seq(100, 1000, length.out = 100)
R <- ricker(S, a = 1.9, b = 900)
plot(S, R)
```
**ricker_escapement**  
*Assign a salmon escapement target based on a Ricker curve*

**Description**

Sets escapement according to Hilborn and Walters (1992) p272, Table 7.2. \( S_{\text{msy}} = b(0.5 - 0.07a) \).

**Usage**

\[
ricker\_escapement(a, b)
\]

**Arguments**

- **a**: Ricker productivity parameter.
- **b**: Ricker density-dependent parameter.

**References**


**Examples**

\[
ricker\_escapement(1.1, 1000)
\]

---

**ricker_v_t**  
*Ricker stock-recruit function with specified error*

**Description**

Ricker stock-recruit function with specified error

**Usage**

\[
ricker\_v\_t(spawners, a, b, d, v\_t)
\]

**Arguments**

- **spawners**: A single spawner abundance
- **a**: Ricker productivity parameter. Recruits are \( e^a \) at the origin.
- **b**: Ricker density dependent parameter.
- **d**: Depensation parameter. A value of 1 means no depensation. Larger values indicate depensation.
- **v\_t**: A single residual on the curve. Will be exponentiated. Note that we are *not* bias correcting within this function (subtracting half the variance squared) and so the deviations will not be mean unbiased unless they were bias corrected previously.
run_cons_plans

Value

Returns a vector of recruits.

Examples

plot(1, 1, xlim = c(1, 100), ylim = c(0, 90), type = "n", xlab = "Spawners", ylab = "Returns")
for(i in 1:100) {
  points(i, ricker_v_t(i, a = 1.1, b = 60, d = 1, v_t = rnorm(1, mean = -(0.1^2)/2, sd = 0.1)))
}

run_cons_plans

Run conservation plans and return the portfolio mean and variance values

Description

This function takes a set of weights representing different conservation plans and gets the mean and variance in portfolio space. This function allows a maximally complicated set of weights to accommodate all possible scenarios. It can accommodate different spatial strategies of conservation, conserving different numbers of populations, and a lack of knowledge. You can do this by how you set your w weight object. See the example.

Usage

run_cons_plans(w, env_type, env_params, show_progress = TRUE, burn = 1:30, assess_freq = 5, risk_fn = var, ...)

Arguments

w A (nested) list of weights. The first list level contains the different plans. The next level contains repetitions for a given plan. E.g. cp[[2]][[1]] contains the first iteration of the second conservation plan. Each end element should be a matrix of weights with one row and the number of columns equal to the number of subpopulations.
env_type The environmental type to pass to generate_env_ts
env_params The environmental parameters to pass to generate_env_ts
show_progress Logical: show an indication of progress?
burn Cycles to throw out as burn in
assess_freq How frequently (in years) to re-assess the Ricker a and b values.
risk_fn A risk function to use. Can be any function that takes a numeric vector and returns a single value. Suggested values include var, or VaR, or CVaR. Defaults to variance.
... Other values to pass to meta_sim
Value

A list with two high-level elements: the mean variance output (plans_mv) and the raw simulation output (plans_port). Within plans_mv, each element of the list contains a conservation plan. Each row of the data frames represents a trial run. Within plans_port, each first level of the list contains a weight element and each second level of the list contains a replicate.

Examples

```r
## Not run:
set.seed(1)
w_plans <- list()
w_plans[[1]] <- c(5, 1000, 5, 1000, 5, 1000, 5, 1000, 5)
w_plans[[2]] <- c(5, 5, 1000, 1000, 1000, 1000, 5, 5)
w_plans[[3]] <- c(rep(1000, 4), rep(5, 6))
w_plans[[4]] <- rev(w_plans[[3]])

plans_name_sp <- c("Full range of responses", "Most stable only", "Lower half", "Upper half")
n_trials <- 50 # number of trials at each n conservation plan
n_plans <- 4 # number of plans
num_pops <- c(2, 4, 8, 16) # n pops to conserve
w <- list()
for(i in 1:n_plans) { # loop over number conserved
  w[[i]] <- list()
  for(j in 1:n_trials) { # loop over trials
    w[[i]][[j]] <- matrix(rep(625, 16), nrow = 1)
    w[[i]][[j]][, sample(1:16, num_pops[i])] <- 5
  }
}

arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)

x_arma_sp <- run_cons_plans(w, env_type = "arma", env_params = arma_env_params)

plot_cons_plans(x_arma_sp, plans_mv, plans_name = plans_name_sp, cols = cols, add_all_efs = FALSE, xlim = c(0.02, 0.15), ylim = c(-0.017, 0.017), add_legend = FALSE)

# In this version, the pops are wiped out; total abundance changes
n_trials <- 50 # number of trials at each n conservation plan
num_pops <- c(2, 4, 8, 16) # n pops to conserve
n_plans <- length(num_pops) # number of plans
w <- list()
for(i in 1:n_plans) { # loop over number conserved
  w[[i]] <- list()
  for(j in 1:n_trials) { # loop over trials
    w[[i]][[j]] <- matrix(rep(1000, 16), nrow = 1)
    w[[i]][[j]][, sample(1:16, num_pops[i])] <- 5
  }
}

plans_name_n <- paste(num_pops, "populations")
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)

x_arma_n <- run_cons_plans(w, env_type = "arma", env_params = arma_env_params)
```

---

```
run_cons_plans
```

---

**Value**

A list with two high-level elements: the mean variance output (plans_mv) and the raw simulation output (plans_port). Within plans_mv, each element of the list contains a conservation plan. Each row of the data frames represents a trial run. Within plans_port, each first level of the list contains a weight element and each second level of the list contains a replicate.

**Examples**

```r
## Not run:
set.seed(1)
w_plans <- list()
w_plans[[1]] <- c(5, 1000, 5, 1000, 5, 1000, 5, 1000, 5)
w_plans[[2]] <- c(5, 5, 1000, 1000, 1000, 1000, 5, 5)
w_plans[[3]] <- c(rep(1000, 4), rep(5, 6))
w_plans[[4]] <- rev(w_plans[[3]])

plans_name_sp <- c("Full range of responses", "Most stable only", "Lower half", "Upper half")
n_trials <- 50 # number of trials at each n conservation plan
n_plans <- 4 # number of plans
num_pops <- c(2, 4, 8, 16) # n pops to conserve
w <- list()
for(i in 1:n_plans) { # loop over number conserved
  w[[i]] <- list()
  for(j in 1:n_trials) { # loop over trials
    w[[i]][[j]] <- matrix(rep(625, 16), nrow = 1)
    w[[i]][[j]][, sample(1:16, num_pops[i])] <- 5
  }
}

arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)

x_arma_sp <- run_cons_plans(w, env_type = "arma", env_params = arma_env_params)

plot_cons_plans(x_arma_sp, plans_mv, plans_name = plans_name_sp, cols = cols, add_all_efs = FALSE, xlim = c(0.02, 0.15), ylim = c(-0.017, 0.017), add_legend = FALSE)

# In this version, the pops are wiped out; total abundance changes
n_trials <- 50 # number of trials at each n conservation plan
num_pops <- c(2, 4, 8, 16) # n pops to conserve
n_plans <- length(num_pops) # number of plans
w <- list()
for(i in 1:n_plans) { # loop over number conserved
  w[[i]] <- list()
  for(j in 1:n_trials) { # loop over trials
    w[[i]][[j]] <- matrix(rep(1000, 16), nrow = 1)
    w[[i]][[j]][, sample(1:16, num_pops[i])] <- 5
  }
}

plans_name_n <- paste(num_pops, "populations")
arma_env_params <- list(mean_value = 16, ar = 0.1, sigma_env = 2, ma = 0)

x_arma_n <- run_cons_plans(w, env_type = "arma", env_params =
thermal_area

arma_env_params, max_a = thermal_integration(16))

plot_cons_plans(x arma_n$plans_mv, plans_name = plans_name_n, cols =
cols, add_all_efs = FALSE, xlim = c(0.02, 0.15), ylim = c(-0.017,
0.017), add_legend = FALSE)

## End(Not run)

thermal_area  Return desired squared deviation between desired area and actual
area under a curve

Description

The function finds the lower and upper roots (where the thermal curve crosses 0) with the uniroot
function and then integrates the area under the thermal curve with the integrate function. This is
useful as part of the optimization routine in optim_thermal.

Usage

thermal_area(max_a, desired_area, optim_temp, width_param, lower = -5,
upper = 40)

Arguments

max_a       Maximum Ricker a productivity value
desired_area Desired area under the thermal curve
optim_temp   Optimal temperature
width_param  The width parameter as a numeric value
lower        Lower bound to pass to uniroot
upper        Upper bound to pass to uniroot

thermal_curve_a  Create thermal tolerance curves.

Description

 Creates a quadratic thermal tolerance curve of the form: width_param * (temp - optim_temp)^2 +
max_a Negative values are *not* returned as 0 for speed of computation. You should check for this
after.

Usage

thermal_curve_a(temp, optim_temp = 15, max_a = 1.4, width_param = 0.02)
**thermal_integration**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>The input temperature value.</td>
</tr>
<tr>
<td>optim_temp</td>
<td>The optimal temperature.</td>
</tr>
<tr>
<td>max_a</td>
<td>The maximum productivity parameter ‘a’ from a Ricker model (or whatever the y-axis value is you want to return).</td>
</tr>
<tr>
<td>width_param</td>
<td>A parameter to control the width of the parabola. Smaller numbers make wider parabolas.</td>
</tr>
</tbody>
</table>

**Value**

A productivity parameter given the location on a thermal tolerance curve.

**Examples**

```r
x <- seq(5, 30, length.out = 200)
plot(x, thermal_curve_a(x), ylab = "a", xlab = "Temperature", type = "l")
```

**Description**

Get maximum Ricker a values for a given number of populations. Useful for assembling multiple thermal tolerance curves in which each has the same total area under it.

**Usage**

```r
thermal_integration(n_pop, width_params = c(seq(0.05, 0.02, length.out = n_pop/2), rev(seq(0.05, 0.02, length.out = n_pop/2))), optim_temps = seq(13, 19, length.out = n_pop), desired_area = 30)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_pop</td>
<td>The number of populations.</td>
</tr>
<tr>
<td>width_params</td>
<td>Desired widths of the thermal tolerance curves.</td>
</tr>
<tr>
<td>optim_temps</td>
<td>Temperature value at which to reach the peak of each thermal tolerance curve.</td>
</tr>
<tr>
<td>desired_area</td>
<td>Desired area under each curve.</td>
</tr>
</tbody>
</table>
Examples

# Minimal example:
thermal_integration(16)

# Elaborate example:
optim_temps <- seq(13, 19, length.out = 10)
widths <- c(seq(0.05, 0.02, length.out = 5), rev(seq(0.05, 0.02, length.out = 5)))
heights <- c(seq(2.8, 2.2, length.out = 5), rev(seq(2.8, 2.2, length.out = 5)))
x <- seq(3, 29, length.out = 200)
plot(1, 1, xlim = c(4, 28), ylim = c(-0.01, 2.9), ylab = "Environmental value", type = "n", yaxs = "i", las = 1)
for(i in 1:10) {
  a <- thermal_curve_a(x, optim_temp = optim_temps[i], max_a = heights[i], width_param = widths[i])
  lines(x, a, col = "grey40", lwd = 1.5)
}

---

**VaR** | **Value at Risk**
--- | ---

Description

Get the value at risk.

Usage

VaR(x, probs = 0.05)

Arguments

- **x** | A numeric vector
- **probs** | The probability cutoff to pass to the value at risk.
# Index

<table>
<thead>
<tr>
<th>Function/Package</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_dens_polygon</td>
<td>3</td>
</tr>
<tr>
<td>annotate</td>
<td>3</td>
</tr>
<tr>
<td>axis</td>
<td>20</td>
</tr>
<tr>
<td>barplot</td>
<td>23</td>
</tr>
<tr>
<td>count_quasi_exts</td>
<td>4</td>
</tr>
<tr>
<td>create_asset_weights</td>
<td>5, 15, 19</td>
</tr>
<tr>
<td>custom_bw</td>
<td>6</td>
</tr>
<tr>
<td>CVaR</td>
<td>6, 29</td>
</tr>
<tr>
<td>est_beta_params</td>
<td>6</td>
</tr>
<tr>
<td>fastlm</td>
<td>7</td>
</tr>
<tr>
<td>fit_ricker</td>
<td>7, 17</td>
</tr>
<tr>
<td>generate_env_ts</td>
<td>8, 15, 17, 29</td>
</tr>
<tr>
<td>generate_straying_matrix</td>
<td>9</td>
</tr>
<tr>
<td>get_conserv_plans_mv</td>
<td>9, 11</td>
</tr>
<tr>
<td>get_efficient_frontier</td>
<td>10</td>
</tr>
<tr>
<td>get_port_vals</td>
<td>11</td>
</tr>
<tr>
<td>get_quantile_contour</td>
<td>11</td>
</tr>
<tr>
<td>gg_color_hue</td>
<td>12</td>
</tr>
<tr>
<td>hcl</td>
<td>12</td>
</tr>
<tr>
<td>impl_error</td>
<td>12, 16, 18</td>
</tr>
<tr>
<td>integrate</td>
<td>31</td>
</tr>
<tr>
<td>is_element</td>
<td>13</td>
</tr>
<tr>
<td>is_quasi_ext</td>
<td>14</td>
</tr>
<tr>
<td>kde2d</td>
<td>12</td>
</tr>
<tr>
<td>legend</td>
<td>21</td>
</tr>
<tr>
<td>matplotlib</td>
<td>24</td>
</tr>
<tr>
<td>meta_sim</td>
<td>4, 10, 11, 14, 15, 16, 19, 22, 24, 26, 29</td>
</tr>
<tr>
<td>metafolio</td>
<td>14</td>
</tr>
<tr>
<td>metafolio package</td>
<td>14</td>
</tr>
<tr>
<td>metasim_base</td>
<td>15</td>
</tr>
<tr>
<td>monte_carlo_portfolios</td>
<td>15, 19, 23</td>
</tr>
<tr>
<td>mtext</td>
<td>25</td>
</tr>
<tr>
<td>my.axis</td>
<td>20</td>
</tr>
<tr>
<td>optim_thermal</td>
<td>20, 31</td>
</tr>
<tr>
<td>plot.default</td>
<td>21, 23, 25, 26</td>
</tr>
<tr>
<td>plot_cons_plans</td>
<td>11, 15, 21</td>
</tr>
<tr>
<td>plot_correlation_between_returns</td>
<td>15, 22</td>
</tr>
<tr>
<td>plot_efficient_portfolios</td>
<td>15, 22</td>
</tr>
<tr>
<td>plot_panel_lines</td>
<td>24</td>
</tr>
<tr>
<td>plot_rickers</td>
<td>15, 24</td>
</tr>
<tr>
<td>plot_sim_ts</td>
<td>14, 25</td>
</tr>
<tr>
<td>plot_sp_a_ts</td>
<td>26</td>
</tr>
<tr>
<td>ricker</td>
<td>27</td>
</tr>
<tr>
<td>ricker_escapement</td>
<td>28</td>
</tr>
<tr>
<td>ricker_v_t</td>
<td>28</td>
</tr>
<tr>
<td>run_cons_plans</td>
<td>15, 21, 29</td>
</tr>
<tr>
<td>text</td>
<td>3</td>
</tr>
<tr>
<td>thermal_area</td>
<td>31</td>
</tr>
<tr>
<td>thermal_curve_a</td>
<td>31</td>
</tr>
<tr>
<td>thermal_integration</td>
<td>17, 32</td>
</tr>
<tr>
<td>unroot</td>
<td>31</td>
</tr>
<tr>
<td>VaR</td>
<td>29, 33</td>
</tr>
</tbody>
</table>