Package: mse (via r-universe)

November 12, 2026

Title Tools for Running Management Strategy Evaluations using FLR

Version 2.4.14

Description A set of functions and methods to enable the development and running of Management Strategy Evaluation (MSE) analyses, using the FLR packages and classes and the a4a methods and algorithms.

X-schema.org-keywords simulation, MSE, fisheries, flr, a4a, R

Depends R(>= 4.0), methods, data.table, FLCore(>= 2.6.22), FLasher(>= 0.7.1), future, progressr,

Imports doFuture, foreach, FLFishery, FLBRP(>= 2.5.8), ggplot2, ggplotFL(>= 2.7.2), ggrepel, patchwork

Additional_repositories https://flr.r-universe.dev

Suggests testthat, knitr, rmarkdown, FLa4a, doParallel

Collate 'generics.R' 'mseCtrl-class.R' 'mpCtrl-class.R' 'FLo-class.R' 'FLom-class.R' 'FLombf-class.R' 'oem.R' 'FLoem-class.R' 'FLiem-class.R' 'FLmse-class.R' 'FLmses-class.R' 'dispatch.R' 'mp.R' 'tune.R' 'tracking.R' 'performance.R' 'plot.R' 'data.R' 'utilities.R' 'sa.R' 'ind.R' 'phcr.R' 'hcr.R' 'is.R' 'iem.R' 'fb.R' 'tm.R' 'statistics.R'

License EUPL

LazyLoad true

LazyData false

BugReports https://github.com/flr/mse/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

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

Date/Publication 2026-11-10 10:52:13 UTC

RemoteUrl https://github.com/flr/mse

RemoteRef HEAD

RemoteSha 847570cadf03c26699497e0831c1631c68f42c84

Contents

cpue.ind . . . 2
cpues.ind . . . 4
debug-mse . . . 6
fixedC.hcr . . . 7
fixedF.hcr . . . 8
FLiem-class . . . 10
FLmse-class . . . 11
FLoem-class . . . 13
FLom . . . 14
FLombf . . . 17
fwd.om . . . 18
hockeystick.hcr . . . 18
hr,FLombf-method . . . 20
index.hat,FLIndexBiomass,FLStock-method . . . 21
indicator.is . . . 21
mp . . . 22
mpCtrl-class . . . 24
mpDispatch . . . 27
mps . . . 28
mseCtrl-class . . . 30
partialHR . . . 32
perfect.oem . . . 33
perfect.sa . . . 34
performance . . . 34
plesim . . . 38
plot_hockeystick.hcr . . . 38
sampling.oem . . . 40
selectMetric . . . 42
setcontrol . . . 43
split.is . . . 43
statistics . . . 44
tac.is . . . 46
tunebisect . . . 48

Index **50**

`cpue.ind` *Computes CPUE-based Indicators of changes in Stock Abundance*

Description

This function computes four abundance indicators from one CPUE or biomass index of abundance: the index itself, an average over a number of years, a weighted mean over those same years and the slope of the trend over the same period.

Usage

```
cpue.ind(stk, idx, index = 1, nyears = 5, args, tracking)
```

Arguments

stk	An object representing the stock returned by the oem modules, FLStock.
idx	An FLIndices containing the indices rturned by the oem module.
index	An integer or character, specifying the index to use from idx. Default is 1.
nyears	An integer, the number of years to consider for the calculations. Default is 5.
args	A list containing dimensionality arguments, passed on by mp().
tracking	An FLQuant used for tracking indicators, intermediate values, and decisions during MP evaluation.

Details

The weighted average returned in the 'wmean' element is calculated over the last nyears. The last year's weight is set as 50%, and the remaining years share the other 50% proportionally. The 'slope' metric is computed on the log-transformed data.

Three elements are added to the tracking table:

- mean.ind, with the index average
- wmean.ind, with the index weighted average
- slope.ind, with the index slope

Value

A list containing 'stk', the input FLStock, 'ind, an FLQuants object containing the computed index metrics, and the 'tracking' table.

Examples

```
data(plesim)
# MP control with CPUE: catch ~ weighted 4-year mean CPUE
ctrl <- mpCtrl(est=mseCtrl(method=cpue.ind, args=list(index=1)),
  hcr=mseCtrl(method=hockeystick.hcr, args=list(metric="wmean",
  trigger=2000, output="catch", target=1.25e5)))

# Run the MP
run <- mp(om, oem, control=ctrl, args=list(iy=2025, fy=2035))

# Plot results
plot(om, run)
```

cpues.ind *Compute a CPUE-based indicator for use in empirical harvest control rules*

Description

cpues.ind computes a relative abundance indicator from one or more CPUE or survey indices (idx), to be used by empirical indicator (ind) modules in an mse Management Strategy Evaluation loop. Indices are first restricted to a recent window of nyears, rescaled by dividing by their mean over a set of reference years (refyrs), and then combined across indices, using weights, with one of three methods: a weighted rescaled mean ("mean"), a weighted mean of standardized z-scores on the log scale ("zscore"), or a weighted mean of smoothed, rescaled indices ("smooth"). For combine = "mean", the function additionally computes the mean level and log-linear slope of the combined indicator over the recent window.

Usage

```
cpues.ind(
  stk,
  idx,
  refyrs,
  nyears = 4,
  indices = names(idx),
  combine = c("mean", "zscore", "smooth"),
  weights = rep(1, length(indices)),
  enp.mult = 0.2,
  args,
  tracking
)
```

Arguments

stk	An FLStock object, returned unmodified in the output list.
idx	An FLIndices object with the CPUE or survey indices from which the indicator is computed.
refyrs	Reference years used to rescale ("mean", "smooth") or standardize ("zscore") each index.
nyears	Number of recent years, ending in dy, over which the indicator is computed. numeric, defaults to 4.
indices	Names of the elements of idx to use. Defaults to names(idx), i.e. all indices in idx.
combine	Method used to combine the rescaled indices into a single indicator, one of "mean", "zscore" or "smooth".
weights	Weights applied to each index, used by all three combine methods, passed on to weighted.mean. Defaults to equal weights, numeric of length length(indices).

enp.mult	Smoothing parameter passed to smooth_index as enp.mult, only used when combine = "smooth". Defaults to 0.2.
args	A list of MSE loop arguments, e.g. ay (assessment year) and dy (last data year), unpacked using mse::spread.
tracking	An mse tracking object (FLQuant) used to store the computed indicator(s) for later inspection.

Details

cpues.ind is designed to be called inside an mse indicator (ind) module, and expects args to contain at least the current assessment year (ay) and last data year (dy), unpacked internally via mse::spread. The mean (and, for combine = "mean", the slope) of the indicator is stored in tracking under "mean.ind" and "slope.ind".

Combination across indices is done by the weighted.mean method for FLQuants, which by default excludes NA values from both the weighted sum and the sum of weights, so indices missing in a given year do not bias the result towards zero or dilute the contribution of the indices that do have data.

Value

A list with three elements:

stk	The input FLStock, unchanged.
ind	An FLQuants with the combined indicator(s): ind, mean and slope for combine = "mean"; mean only for combine = "zscore" or combine = "smooth".
tracking	The updated tracking object.

Author(s)

Iago Mosqueira (WMR), FLR Team.

See Also

mp, weighted.mean, smooth_index, zscore

Examples

```
## Not run:
data(ple4)
data(ple4.indices)

args <- list(ay=2017, dy=2017)
tracking <- FLQuant()

out <- cpues.ind(stk=ple4, idx=ple4.indices, refyrs=2005:2010,
  nyears=4, combine="mean", weights=c(1, 2), args=args,
  tracking=tracking)

## End(Not run)
```

debug-mse *Debugging mse modules*

Description

Set and unset the debugging flag of a function inside the *method* slot of a mseCtrl object.

Usage

```
debug(fun, text = "", condition = NULL, signature = NULL)

## S4 method for signature 'mseCtrl,missing'
debug(fun)

## S4 method for signature 'mseCtrl,missing'
undebug(fun)

## S4 method for signature 'mpCtrl,character'
undebug(fun, signature = NULL)

## S4 method for signature 'mpCtrl,missing'
undebug(fun)

## S4 method for signature 'FLo,ANY'
debug(fun)

## S4 method for signature 'FLo,ANY'
undebug(fun)
```

Arguments

fun	Module or control object to debug.
text	Name of module in mpCtrl.
condition	Unused.
signature	Name of module in mpCtrl.

Details

Modules in the mse control object contain the function to be called in the *method* slot. To debug and check the behaviour of an individual function, the *debug* method will start a browser session next time it is called. Debugging functions requires the parallel flag to be set to FALSE, or that no parallel backend is loaded.

Calling *undebug* on an mpCtrl without specifying a module will check for the debugging status of each of them, and undebug if TRUE.

For objects of classes *FLom* and *FLombf*, *debug* and *undebug* will set and unset the debugging flag on the function stored in the *projection* slot.

Value

Both functions invisibly return NULL

Author(s)

Iago Mosqueira (WMR)

See Also

debug

`fixedC.hcr` *Fixed Catch Harvest Control Rule*

Description

A simple harvest control rule that sets total catch to a fixed target value regardless of stock status or other indicators.

Usage

```
fixedC.hcr(stk, ctrg, args, tracking)
```

Arguments

`stk`	The 'oem' observation or SA estimation, an FLStock object.
`ctrg`	The target catch value, numeric or FLQuant. If numeric, the value is replicated across all iterations and management years.
`args`	A list containing dimensionality arguments, passed on by mp(). Must include elements: ay (advice year), it (number of iterations), mys (management years vector).
`tracking`	An FLQuant used for tracking indicators, intermediate values, and decisions during MP evaluation.

Details

This HCR implements a constant catch strategy where the target catch is directly applied. The function is useful for testing fixed harvest strategies or as a baseline reference in MSE comparisons. The target catch can be specified as either a single numeric value (applied across all iterations and years) or as an FLQuant object with possible year and iteration structure.

Value

A list containing elements 'ctrl' (a fwdControl object) and 'tracking'.

Examples

```
# Example dataset
data(plesim)

# Sets up an mpCtrl for fixed catch at MSY level
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=fixedC.hcr, args=list(ctrg=1500)))

# Runs mp between 2021 and 2035
run <- mp(om, control=ctrl, args=list(iy=2021, fy=2035))

# Plot results
plot(om, run)

# Check output
catch(run)

# Using FLQuant for time-varying catch
ctrg_var <- FLQuant(seq(1000, 2000, length=10), dimnames=list(year=2021:2030))
ctrl2 <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=fixedC.hcr, args=list(ctrg=ctrg_var)))

# Example using function directly
fixedC.hcr(stock(om), ctrg=1500, args=list(ay=2017, it=100,
  mys=2018:2020), tracking=FLQuant())
```

---

fixedF.hcr *Fixed Fishing Mortality Harvest Control Rule*

Description

A simple harvest control rule that sets fishing mortality (F) to a fixed target value regardless of stock status or other indicators.

Usage

```
fixedF.hcr(
  stk,
  ftrg,
  ages = range(stock(stk))[c("minfbar", "maxfbar")],
  args,
  tracking
)
```

Arguments

stk	The 'oem' observation or SA estimation, an FLStock object.
ftrg	The target fishing mortality value, numeric or FLQuant. If numeric, the value is replicated across all iterations and management years.

ages	A vector of length 2 specifying the minimum and maximum ages over which F is averaged (fbar range), numeric. Defaults to minfbar and maxfbar from the *stk* FLStock range.
args	A list containing dimensionality arguments, passed on by mp(). Must include elements: ay (advice year), it (number of iterations), mys (management years vector).
tracking	An FLQuant used for tracking indicators, intermediate values, and decisions during MP evaluation.

Details

This HCR implements a constant fishing mortality strategy where the target F is directly applied. The function is useful for testing fixed harvest strategies or as a baseline reference in MSE comparisons. The target F can be specified as either a single numeric value (applied across all iterations and years) or as an FLQuant object with possible year and iteration structure.

Value

A list containing elements 'ctrl' (a fwdControl object) and 'tracking'.

Examples

```
# Example dataset
data(plesim)

# Sets up an mpCtrl for fixed F at Fmsy
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=fixedF.hcr, args=list(ftrg=0.18)))

# Runs mp between 2021 and 2035
run <- mp(om, control=ctrl, args=list(iy=2021, fy=2035))

# Plot results
plot(om, run)

# Check output
fbar(run)

# Using FLQuant for time-varying F
ftrg_var <- FLQuant(seq(0.1, 0.3, length=10), dimnames=list(year=2021:2030))
ctrl2 <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=fixedF.hcr, args=list(ftrg=ftrg_var)))

# Example using function directly
fixedF.hcr(stock(om), ftrg=0.13, args=list(ay=2017, it=100,
  mys=2018:2020), tracking=FLQuant())
```

FLiem-class *S4 class* FLiem

Description

The FLiem class stores the information relative to the implementation error model of the MSE.

Usage

```
## S4 method for signature 'FLiem'
initialize(.Object, ...)

## S4 method for signature 'FLiem'
iter(obj, i)
```

Arguments

...	additional argument list that might never be used
object	object of relevant class (see signature of method)

Slots

method The method to berun, class function.

args Arguments to be used when method is called, class list.

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All slots are then created to match the requirements of the class validity. If an unnamed FLQuant object is provided, this is used for sizing, but not for populating any slot.

FLmse-class *S4 class* FLmse

Description

FLmse stores the operating model, tracking object, control (mpCtrl), and oem as used during an MSE run as well as miscellaneous args. Accessors are provided for om, tracking, control and oem.

The FLmse class stores information relative to the MSE's management procedure'.

Usage

```
FLmse(...)

FLmse(...)

om(object, ...)

## S4 method for signature 'FLmse'
om(object)

om(object) <- value

tracking(object, ...)

## S4 method for signature 'FLmse'
tracking(object, biol = "missing")

tracking(object, ...) <- value

## S4 method for signature 'FLmse'
control(object, i = NULL)

oem(object, ...)

## S4 method for signature 'FLmse'
oem(object)

oem(object) <- value

## S4 method for signature 'FLmse'
args(name)

## S4 replacement method for signature 'FLmse,list'
args(object) <- value
```

Arguments

`...`	additional argument list that might never be used
`object`	object of relevant class (see signature of method)
`value`	the new object

Details

FLmse class

Representation of a single Management Strategy Evaluation (MSE) projection.

Methods provided on FLmse dispatch to slots in the contained OM and OEM (e.g. stock(), catch(), ssb(), fbar(), metrics()). Use iter() to change iteration slices for contained objects.

Slots

om FLo. The operating model used by the simulation.

tracking FLQuants. Tracking structure used to record decisions and diagnostics.

control mpCtrl. Management procedure control used for the run.

oem FLoem. Observation error model configuration.

args list. Miscellaneous MSE arguments (iy, fy, ay, dy, data_lag, management_lag, etc.).

om `FLom` with the operating model.

tracking `FLQuant` with record of decisions made during the mp cycle.

args `list` with assorted arguments required to run the MSE cycle.

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All slots are then created to match the requirements of the class validity. If an unnamed `FLQuant` object is provided, this is used for sizing, but not for populating any slot.

Examples

```
## Not run:
# construct minimal FLmse
fm <- methods::new("FLmse")
om(fm) <- methods::new("FLo")

## End(Not run)
```

FLoem-class *Specification for the observation error model (OEM).*

Description

The FLoem class stores the method, arguments and observations that define the way observations are collected from an operating model at each time step in the management procedure. This class extends *mseCtrl* through the addition of two lists used to gather past and new observations, and deviances to use on each step in the observation process.

Usage

```
FLoem(...)

FLoem(...)

observations(object, ...)

## S4 method for signature 'FLoem'
observations(object, ...)

observations(object, i) <- value

## S4 replacement method for signature 'FLoem,missing,list'
observations(object) <- value

## S4 replacement method for signature 'FLoem,ANY,FLStock'
observations(object, i) <- value

## S4 method for signature 'FLoem'
deviances(object, ...)

## S4 replacement method for signature 'FLoem,list'
deviances(object, ...) <- value

## S4 method for signature 'FLoem'
show(object)

## S4 method for signature 'FLoem'
iter(obj, iter)

## S4 method for signature 'FLoem,FLoem'
combine(x, y, ..., check = FALSE)
```

Arguments

...	additional argument list that might never be used

object	object of relevant class (see signature of method)
value	the new object

Slots

method The function to be run in the module call, class *function*.

args Arguments to be passed to the method, of class *list*.

observations Past observations, class *list*.

deviances Observation deviances, class *list*.

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All slots are then created to match the requirements of the class validity. If an unnamed FLQuant object is provided, this is used for sizing, but not for populating any slot.

Examples

```
data(plesim)
deviances(oem)
deviances(oem, "stk")
deviances(oem, "stk", "catch.n")
data(plesim)
x <- iter(oem, 1:50)
dims(observations(x)$stk)$iter
dims(deviances(x)$stk$catch.n)$iter
data(plesim)
iter(oem, 1:50)
```

FLom *A class for an operating model (OM)*

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque eleifend odio ac rutrum luctus. Aenean placerat porttitor commodo. Pellentesque eget porta libero. Pellentesque molestie mi sed orci feugiat, non mollis enim tristique. Suspendisse eu sapien vitae arcu lobortis ultrices vitae ac velit. Curabitur id

Usage

```
FLom(...)

FLom(...)

## S4 method for signature 'FLom'
stock(object)

## S4 replacement method for signature 'FLom,FLStock'
stock(object) <- value

## S4 method for signature 'FLom'
sr(object)

## S4 replacement method for signature 'FLom,FLSR'
sr(object) <- value

## S4 method for signature 'FLom,FLom'
combine(x, y, ...)

## S4 replacement method for signature 'FLmse,FLo'
om(object) <- value

## S4 replacement method for signature 'FLmse,data.table'
tracking(object) <- value

## S4 replacement method for signature 'FLmse,mpCtrl'
control(object) <- value

## S4 replacement method for signature 'FLmse,FLoem'
oem(object) <- value
```

Arguments

...	additional argument list that might never be used
object	object of relevant class (see signature of method)
value	Object to assign in slot

Slots

`stock` The population and catch history, `FLStock`.

`sr` The stock-recruitment relationship, `FLSR`.

`refpts` The estimated reference points, `FLPar`.

`fleetBehaviour` Dynamics of the fishing fleet to be used in projections, `mseCtrl`.

Validity

stock and sr dimensions Dimensions 2:6 of the `stock` and `sr` slots must match.

rec age Stock and stock recruitment residuals must use the recruitment age.

You can inspect the class validity function by using `getValidity(getClassDef('FLom'))`

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All unspecified slots are then created to match the requirements of the class validity function.

Methods

Methods exist for various calculations based on values stored in the class:

METHOD Neque porro quisquam est qui dolorem ipsum.

Author(s)

The FLR Team

See Also

FLComp

Examples

```
data(plesim)
comb <- combine(iter(om, 1:50), iter(om, 51:100))
all.equal(om, comb)
```

FLombf *Append two FLombf objects along the year dimension*

Description

Joins two FLombf objects along the year axis: the first object is windowed to after, and the second from after + 1 to its last year. The two windowed pieces are then combined using FLCore::append on the underlying FLBiols and FLFisheries slots.

Usage

```
FLombf(...)

## S4 method for signature 'FLombf,FLombf'
append(x, values, after = dims(values)$minyear)
```

Arguments

...	additional argument list that might never be used
x	An FLombf object, used up to year after.
values	An FLombf object, used from year after + 1.
after	Integer year. The last year taken from x. Defaults to the last year of x.
object	object of relevant class (see signature of method)

Value

A new FLombf object spanning the years of x up to after joined with the years of values from after + 1.

Examples

```
data(plesim)
# Split an FLombf at year 2020 and re-join
om1 <- window(om, end=2020)
om2 <- window(om, start=2021)
om3 <- append(om1, om2, after=2020)
```

`fwd.om` *A method to project the operating model (OM)*

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque eleifend odio ac rutrum luctus. Aenean placerat porttitor commodo. Pellentesque eget porta libero. Pellentesque molestie mi sed orci feugiat, non mollis enim tristique. Suspendisse eu sapien vitae arcu lobortis ultrices vitae ac velit. Curabitur id

Usage

```
fwd.om(om, ctrl, ...)
```

Arguments

`ctrl`	the fwdControl object with objectives and constraints
`object`	the OM as a FLStock

`hockeystick.hcr` *Hockey-stick Harvest Control Rule*

Description

A hockey-stick harvest control rule that sets the value of an output between a minimum and a target, according to that of a metric, compared with a limit and a trigger.

Usage

```
hockeystick.hcr(
  stk,
  ind,
  target,
  trigger,
  lim = 0,
  min = 0,
  drop = 0,
  metric = "ssb",
  output = "fbar",
  dlow = NULL,
  dupp = NULL,
  all = TRUE,
  initial = NULL,
  args,
  tracking,
  ...
)
```

Arguments

stk	The 'oem' observation or SA estimation, an FLStock object.
ind	Possible indicators returned by the 'est' step, FLQuants.
target	The output level applied when the stock metric is at or above the trigger value, numeric or FLQuant.
trigger	The stock status or metric value threshold above which the control variable reaches the target value, numeric or FLQuant.
lim	The lower threshold of the stock metric, below which the control rule applies the minimum allowable value (min), numeric or FLQuant.
min	Numeric. The minimum allowable control value (e.g., minimum F or catch).
drop	Numeric. A stock metric threshold below which the control variable is forced to min to prevent over-exploitation of severely depleted stocks, numeric or FLQuant
metric	The stock metric to use for the HCR (e.g., "ssb" for spawning-stock biomass), character or function.
output	Character. The output control variable (e.g., "fbar" for fishing mortality or "catch" for quotas), character.
dlow	A limit for the decrease in the output variable, e.g. 0.85 for a maximum decrease of 15%, numeric. No limit is applied if NULL.
dupp	A limit for the increase in the output variable, e.g. 1.15 for a maximum increase of 15%, numeric. No limit is applied if NULL.
all	If TRUE, upper and lower limits (dupp and dlow) are applied unconditionally, otherwise only when metric > trigger, logical.
args	A list containing dimensionality arguments, passed on by mp().
tracking	An FLQuant used for tracking indicators, intermediate values, and decisions during MP evaluation.
initial.	Initial value of 'output' to use when applying 'dlow' and 'dupp' limits, numeric of FLQuant.

Details

This function implements a hockey-stick shaped harvest control rule (HCR). It is commonly used to set F or effort, and sometimes catch, from a measure of stock status, such as estimated SSB or depletion. The function can take any input, 'metric', if it can computed from an FLStock or is returned by the function run in the 'est' step. The 'output' argument can be set to any quantity that the OM projection method can understand, e.g. fbar, effort or catch for 'fwd.om'. The HCR increases the control variable linearly between a lower threshold (lim) and a trigger point (trigger), and sets it to a target level (target) when the metric is above the trigger. The decreasing line can be cut at any point ('drop'), where output values fall to the 'min' level set. See examples below. The function can also apply optional upper (dupp) and lower (dlow) constraints to changes in the control variable. These constraints can be applied either unconditionally (all = TRUE) or only on the stock being above the trigger.

Value

A list containing elements 'ctrl', a fwdControl object, and 'tracking'. Three elements in *tracking* report on the steps inside *hockeystick.hcr*:

- decision.hcr

Examples

```
# Example dataset
data(plesim)

# Sets up an mpCtrl using hockeystick(fbar~ssb)
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(metric="ssb", trigger=14000,
    output="fbar", target=0.18)))

plot_hockeystick.hcr(ctrl)

# Sets up a 40:10 HCR mpCtrl using hockeystick(fbar~depletion)
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(
    metric="depletion", trigger=0.40, lim=0.10,
    output="fbar", target=0.18, min=0.02)))

plot_hockeystick.hcr(ctrl)

# Runs mp between 2021 and 2035
run <- mp(om, control=ctrl, args=list(iy=2021, fy=2035))

# Plots results
plot(om, run)
```

`hr,FLombf-method` *hr method for FLombf*

Description

S4 method for `hr(object)` when `object` is an `FLombf`. The method sums partial harvest-rate contributions across fisheries to produce a total harvest-rate estimate.

Usage

```
## S4 method for signature 'FLombf'
hr(object)
```

Arguments

`object`	An `FLombf` object.

Value

An FLQuant (or aggregated FLQuant) that is the sum of the partial harvest rates across fisheries.

---

index.hat,FLIndexBiomass,FLStock-method *Predicted index of abundance from abundance estimates*

Description

Predicted index of abundance from abundance estimates

Usage

```
## S4 method for signature 'FLIndexBiomass,FLStock'
index.hat(object, stock)
```

Arguments

object	An FLIndexBiomass object.
stock	An FLStock object with estimated abundances, *stock.n*.

Value

An FLQuant for the precited index of abundance in biomass.

---

indicator.is *Indicator Implementation System Module*

Description

Applies the harvest control rule (HCR) decision to scale the target catch or fishing mortality based on a reference quantity (status-quo fishing mortality or catch).

Usage

```
indicator.is(stk, ctrl, args, tracking, system = c("output", "input"), ...)
```

Arguments

stk	The perceived FLStock object returned by the OEM module.
ctrl	The fwdControl object output by the *hcr* step, containing the HCR decision.
args	The MSE run arguments, including sqy (status-quo years).
tracking	The tracking object for recording module decisions and outputs.
system	Character, either "output" (default) or "input". If "output", catch is used as reference; if "input", fishing mortality is used.
...	Additional arguments (currently unused).

Details

This implementation system (IS) function adjusts the management decision from the HCR step by multiplying it with a reference quantity computed from the stock-at-age data in the perceived stock, allowing for indicator-based management approaches. The reference quantity can be either the status-quo fishing mortality (input system) or catch (output system).

When `system="output"`, the function scales the HCR target by the mean catch from the status-quo years. When `system="input"`, it scales by the mean fishing mortality. This approach allows MPs to work with relative rather than absolute targets.

Value

A list containing:

`ctrl`	The modified fwdControl with scaled target values.
`tracking`	The updated tracking object.

Author(s)

Ernesto Jardim, Iago Mosqueira

See Also

`tac.is`, `sp.is`, `seasonal.is`

Examples

```
data(plesim)
# Setup control with indicator.is using output system (catch-based)
control <- mpCtrl(list(est=mseCtrl(method=perfect.sa),
  hcr=mseCtrl(method=hockeystick.hcr,
    args=list(lim=0, trigger=0.5, target=1.2)),
  isys=mseCtrl(method=indicator.is, args=list(system="output"))))
# Run MP
run <- mp(om, oem, control=control, args=list(iy=2021, fy=2027))
```

---

mp *mp executes a single run of a Management Procedure*

Description

An individual management procedure (MP) defined by the `control` argument, is run under a temporal configuration defined in the `args` list, and on a given operating model (`om`). Particular observation error (`oem`), and implementation error model (`iem`) elements can also be specified.

Usage

```
mp(
  om,
  oem = NULL,
  iem = NULL,
  control = ctrl,
  ctrl = control,
  args,
  scenario = "NA",
  tracking = "missing",
  verbose = !handlers(global = NA),
  progress = handlers(global = NA),
  parallel = TRUE,
  window = TRUE,
  .DEBUG = FALSE
)
```

Arguments

om	The operating model (OM), an object of class *FLom* or *FLombf*.
oem	The observation error model (OEM), an object of class *FLoem*.
iem	The implementation error model (IEM), an object of class *FLiem*.
ctrl	A control structure for the MP run, an object of class *mpCtrl*.
args	MSE arguments, *list*. Only 'iy', the intermediate (starting) year, is required.
scenario	Name of the scenario tested in this run, *character*.
tracking	Extra elements (rows) to add to the standard tracking *FLQuant* in its first dimensions, *character*. Elements can also be added by individual control modules.
verbose	Should output be verbose (year being executed) or not, *logical*.

Details

Calls to `mp()` can be run in parallel using `%dofuture%`. Iterations in the `om` are split across the number of availabnle workers, as set by a call to `plan()`. This can specially improve computations for MPs fitting any kind of model or doing other non-vectorizsed calculations.

Progress in the simulation is reported via a progress bar as set by the `progressr` package, if a global handler is set up using `handlers(global=TRUE)`. This bar works when a parallel plan has been set up. Otherwise, a simple print out of the year being run is shown.

The `args` list controls the timing of the simulation and its elements:

- iy: the initial year of simulation inn which decisions are made. The only required element in `args`.
- fy: final year, defaults to last year in `om` object.
- y0: first data year, defaults to first year in `om` object.
- nsqy: number of years for status-quo calculations, defaults to 3.
- data_lag: number of years between last data point and decision year, defaults to 1.

- management_lag: number of years between decision and its application. Must be greater than 0, and defaults to 1.
- frq: frequendcy of advice in years, defaults to 1.
- vy: vector of years in which advice is givenm, defaults to a sequence between `iy` and `fy` every `frq` years.

Value

An object of class *FLmse*, trimmed to start in year `iy` unless `window` is set to`FALSE`.

Examples

```
# dataset contains both OM (FLom) and OEM (FLoem)
data(plesim)
# Set control: sa and hcr
control <- mpCtrl(list(
  est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(lim=0,
  trigger=14000, target=0.18))))
# Runs mp
tes <- mp(om, oem=oem, ctrl=control, args=list(iy=2021, fy=2034))
# Runs mp with triannual management
tes3 <- mp(om, oem=oem, ctrl=control, args=list(iy=2021, fy=2034, frq=3))
# Compare both runs
plot(om, list(annual=tes, triannual=tes3))
# 'perfect.oem' is used if none is given
tes <- mp(om, ctrl=control, args=list(iy=2021, fy=2035))
plot(om, tes)
```

---

mpCtrl-class *S4 class* mpCtrl

Description

The `mpCtrl` class defines which modules will be run my a call to the `mp` function. It contains a series of objects of class *mseCtrl* only for those modules required by the defined MP.

Usage

```
## S4 method for signature 'mpCtrl'
initialize(.Object, ...)

## S4 method for signature 'mpCtrl'
est(object)

est(object) <- value

## S4 replacement method for signature 'mpCtrl,function'
```

```
est(object) <- value

## S4 replacement method for signature 'mpCtrl,mseCtrl'
est(object) <- value

## S4 replacement method for signature 'mpCtrl,list'
est(object) <- value

## S4 method for signature 'mpCtrl'
phcr(object)

phcr(object) <- value

## S4 replacement method for signature 'mpCtrl,function'
phcr(object) <- value

## S4 replacement method for signature 'mpCtrl,mseCtrl'
phcr(object) <- value

## S4 replacement method for signature 'mpCtrl,list'
phcr(object) <- value

## S4 method for signature 'mpCtrl'
hcr(object)

hcr(object) <- value

## S4 replacement method for signature 'mpCtrl,function'
hcr(object) <- value

## S4 replacement method for signature 'mpCtrl,mseCtrl'
hcr(object) <- value

## S4 replacement method for signature 'mpCtrl,list'
hcr(object) <- value

## S4 method for signature 'mpCtrl'
isys(object)

isys(object) <- value

## S4 replacement method for signature 'mpCtrl,function'
isys(object) <- value

## S4 replacement method for signature 'mpCtrl,mseCtrl'
isys(object) <- value

## S4 replacement method for signature 'mpCtrl,list'
```

```
isys(object) <- value

## S4 method for signature 'mpCtrl'
tm(object)

tm(object) <- value

## S4 replacement method for signature 'mpCtrl,function'
tm(object) <- value

## S4 replacement method for signature 'mpCtrl,mseCtrl'
tm(object) <- value

## S4 replacement method for signature 'mpCtrl,list'
tm(object) <- value

## S4 method for signature 'mpCtrl'
show(object)

## S4 method for signature 'mpCtrl'
iters(object, i)

## S4 method for signature 'mpCtrl'
iter(obj, i)

## S4 method for signature 'mpCtrl'
method(object, element)

## S4 method for signature 'mpCtrl'
args(name)

## S4 replacement method for signature 'mpCtrl,function'
method(object, element) <- value

## S4 replacement method for signature 'mpCtrl,ANY'
args(object, element) <- value

## S4 method for signature 'mpCtrl,character'
debug(fun, text)
```

Arguments

`...`	additional argument list that might never be used
`object`	object of relevant class (see signature of method)

Slots

`est` Specification for the stock status estimator, class *mseCtrl*.

`phcr` Specification for the harvest control rule parametrization, class *mseCtrl*.

`hcr` Specification for the harvest control rule, class *mseCtrl*.

`isys` Specification for the implementation system, class *mseCtrl*.

`tm` Specification for technical measures, class *mseCtrl*.

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All slots are then created to match the requirements of the class validity. If an unnamed `FLQuant` object is provided, this is used for sizing, but not for populating any slot.

Examples

```
mpCtrl(list(
  est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(lim=0,
  trigger=41500, target=0.27))))
mpCtrl(list(
  est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(lim=0,
  trigger=41500, target=0.27))))
```

`mpDispatch` *Title*

Description

Title

Usage

```
mpDispatch(ioval, ..., step)
```

mps *Run Multiple Management Procedure Scenarios*

Description

Executes multiple runs of a Management Procedure (MP) by iterating over a grid of values for a single module's arguments. This allows systematic exploration of parameter sensitivity or scenario comparisons, optionally in parallel. When no extra options are provided, a single MP run is returned as an FLmses object. When options are given, results are either collected as an FLmses or, if statistics are provided, combined into a single data.table of performance statistics.

Usage

```
mps(
  om,
  oem = NULL,
  iem = NULL,
  control = ctrl,
  ctrl = control,
  args,
  statistics = NULL,
  metrics = NULL,
  type = character(1),
  names = NULL,
  progress = handlers(global = NA),
  parallel = TRUE,
  perf = !is.null(statistics),
  ...
)
```

Arguments

om	An FLom object representing the Operating Model.
oem	An FLoem object for the Observation Error Model. Defaults to NULL, in which case perfect.oem is used internally by mp().
iem	An FLiem object for the Implementation Error Model. Defaults to NULL.
control	An mpCtrl object defining the MP modules and their arguments. Can also be passed as ctrl.
ctrl	Alias for control; either may be used.
args	A list of MSE run arguments, including at minimum iy (the initial projection year). May also contain seed to set a random seed for reproducibility.
statistics	A list of performance statistic functions to be passed to performance(). Required when perf = TRUE.
metrics	A list of metric functions passed to performance(). Defaults to NULL.

type	Character string specifying the type of performance output. Passed to performance(). Defaults to character(1).
names	Optional character vector of names for the runs. If a single string is given, it is prepended to the auto-generated names. If NULL, names are constructed automatically from the module name, argument name(s), and rounded values.
parallel	Logical. Should individual MP runs be parallelised using future/doFuture? Defaults to TRUE. Ignored if only one worker is available.
perf	Logical. Should performance statistics be computed and returned instead of the full FLmse objects? Defaults to TRUE when statistics is not NULL.
...	A single named argument corresponding to a module present in control, whose value is a named list of argument vectors to iterate over. Only one module may be varied per call (e.g. hcr = list(Ftarget = c(0.2, 0.3, 0.4))). Passing more than one named element will raise an error.

Details

The function iterates over all combinations of the supplied module argument values. Arguments of unequal length are recycled to the length of the longest. Auto-generated run names take the form <module>_<arg>_<value> for a single argument, or <arg1>-<arg2>_<i> for multiple arguments.

Parallel execution relies on nbrOfWorkers and doFuture::%dofuture%. Progress is reported either per-MP or per-iteration depending on whether the number of runs exceeds the number of workers.

Runs that fail are excluded from the output with a warning; if all runs fail, an error is raised.

Value

If perf = FALSE (or statistics = NULL), an FLmses object containing one FLmse result per parameter set. If perf = TRUE, a single data.table combining performance statistics across all runs, with additional columns for the varied argument values and an mp identifier column.

Author(s)

Iago Mosqueira (WMR)

See Also

mp, performance, mpCtrl, FLmses

Examples

```
## Not run:
data(plesim)
control <- mpCtrl(list(
  est = mseCtrl(method = perfect.sa),
  hcr = mseCtrl(method = catchSSB.hcr, args = list(Ftarget = 0.3, Btrigger = 0))
))

# Run over a grid of Ftarget values, returning performance statistics
res <- mps(om, oem = oem, control = control, args = list(iy = 2021),
```

```
  statistics = statistics,
  hcr = list(Ftarget = c(0.2, 0.3, 0.4)))

# Run without performance statistics, returning FLmses
res <- mps(om, oem = oem, control = control, args = list(iy = 2021),
  hcr = list(Ftarget = c(0.2, 0.3, 0.4)))

## End(Not run)
```

---

mseCtrl-class *S4 class* mseCtrl

Description

The mseCtrl class stores information about how a specific module will be run. The function contained in the *method* slot will be called with three sets of argument: those contained in the *args* slot, the *args* argument to the call to the *mp* function, and the inputs defined by type of module being defined by a particular object. Please see the "Module dispatch" section in the *mse* Technical Manual.

Usage

```
## S4 method for signature 'mseCtrl'
initialize(.Object, ..., method, args)

method(object, ...)

## S4 method for signature 'mseCtrl'
method(object)

method(object, ...) <- value

## S4 replacement method for signature 'mseCtrl,function'
method(object) <- value

args(name)

## S4 method for signature 'mseCtrl'
args(name)

args(object, ...) <- value

## S4 replacement method for signature 'mseCtrl,list'
args(object) <- value

## S4 method for signature 'mseCtrl'
show(object)
```

```
## S4 method for signature 'mseCtrl'
summary(object)

exists(
  x,
  where = -1,
  envir = if (missing(frame)) as.environment(where) else sys.frame(frame),
  frame,
  mode = "any",
  inherits = TRUE
)

## S4 method for signature 'mseCtrl'
exists(x)

exists(
  x,
  where = -1,
  envir = if (missing(frame)) as.environment(where) else sys.frame(frame),
  frame,
  mode = "any",
  inherits = TRUE
)
```

Arguments

`...`	additional argument list that might never be used
`object`	object of relevant class (see signature of method)

Slots

`method` The function to be run in the module call, class *function*.

`args` Arguments to be passed to the method, of class *list*.

Accessors

All slots in the class have accessor and replacement methods defined that allow retrieving and substituting individual slots.

The values passed for replacement need to be of the class of that slot. A numeric vector can also be used when replacing FLQuant slots, and the vector will be used to substitute the values in the slot, but not its other attributes.

Constructor

A construction method exists for this class that can take named arguments for any of its slots. All slots are then created to match the requirements of the class validity. If an unnamed `FLQuant` object is provided, this is used for sizing, but not for populating any slot.

Examples

```
ctl <- mseCtrl(method=function(stk, args, alpha) ssb(stk) * alpha,
  args=list(alpha=0.5))
ctl
method(ctl)
method(ctl) <- function(stk, args, beta) ssb(stk) * beta
args(ctl)
args(ctl) <- list(beta=0.9)
exists(ctl)
```

`partialHR` *hr*

Description

Compute partial harvest-rate estimates for each fishery in a FLombf operating model objecta. For each fishery the function calculates the ratio: catch (by fishery) / sum(S_f * N * W) where S_f is the fishery selectivity (catch.sel), N is numbers-at-age and W are weights at age in the population. The resulting object(s) are given units 'hr'.

Usage

```
partialHR(om)
```

Arguments

`om`	An operating model object (typically an `FLombf` or other object for which `fisheries(om)` and `biol(om)` are defined).

Details

Compute partial harvest rates by fishery

The function sums catches across sexes/other dimensions using `unitSums` and similarly aggregates the product of numbers, weights and selection pattern. The returned values are per-iteration and per-unit as produced by the underlying FL* methods.

Value

A list of FLQuant-like objects (one per fishery) containing the partial harvest rate with units set to 'hr'.

perfect.oem *A perfect observation of catch and abundances-at-age.*

Description

This observation error model function generates a set of perfect observations on catches, biology and abundance. Direct observations are made of the stock, while a single age-structured index of abundance, in numbers, is created with a fixed catchability of 0.01. *deviances* on either *stk$catch.n* or *idx$index*, if given, are applied.

Usage

```
perfect.oem(stk, deviances, observations, args, tracking, biomass = FALSE, ...)
```

Arguments

deviances	A named list of observation deviances, class *list*.
observations	A list of past observations, extended to the end of *om*, class *list*.
args	The mp dimensions arguments, as generated by mp, class *list*.
tracking	Object to track module decisions and outputs, class *FLQuant*.
om	An operating model, class *FLom* or *FLombf*.

Details

This *oem* function generates a full observation time series every time step, and does not append them to existing objects in *observations*.

Value

A named *list* with elements *stk* (*FLStock*), *idx* (*FLIndices*), *deviances*, *observations* and *tracking*.

Examples

```
# On FLom
data(plesim)
obs <- perfect.oem(stock(om), deviances=NULL, observations=NULL,
  args=list(y0=1960, dy=2021), tracking=FLQuant())
```

perfect.sa *A perfect 'estimate' of abundances, catches and harvest.*

Description

The *FLStock* generated by the call to *oem* is simply passed on in this function. The estimates of abundance, catches and exploitation will thus be as precise as the OEM observation.

Usage

```
perfect.sa(stk, idx, args, tracking, ...)
```

Arguments

stk	The stock observation generated by *oem*. Class *FLStock*.
idx	An observation of chnages in abundance, not used. Class *FLIndices*.
args	MSE arguments, class *list*.
tracking	Structure for tracking modules outputs.

Value

A *list* with elements *stk* and *tracking*.

Examples

```
# Example dataset
data(plesim)

# Sets up an mpCtrl for catch ~ MSY
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=fixedC.hcr, args=list(ctrg=11400)))

# Runs mp between 2021 and 2035
run <- mp(om, control=ctrl, args=list(iy=2021, fy=2035))
```

performance *Compute Performance Statistics for Management Procedure Evaluation*

Description

Evaluates the performance of a management procedure by computing statistical metrics across simulated projections. Supports multiple input types (FLQuants, FLStock, FLStocks, FLom, FLmse, FLmses, and lists) and computes custom statistics defined by formulas that reference metrics and reference points.

Usage

```
## S4 method for signature 'FLQuants'
performance(
  x,
  statistics = mse::statistics[c("C", "F", "SB", "AAVC")],
  refpts = FLPar(),
  years = setNames(nm = dimnames(x[[1]])$year[-1]),
  om = character(0),
  type = character(0),
  run = character(0),
  mp = paste(c(om, type, run), collapse = "_"),
  ...
)

## S4 method for signature 'FLo'
performance(
  x,
  refpts = x@refpts,
  statistics = mse::statistics[c("C", "F", "HR", "SB")],
  metrics = NULL,
  om = name(x),
  ...
)

## S4 method for signature 'FLombf'
performance(
  x,
  refpts = FLCore::refpts(x),
  statistics = mse::statistics[c("C", "F", "HR", "SB")],
  metrics = NULL,
  om = name(x),
  ...
)

## S4 method for signature 'FLmse'
performance(
  x,
  refpts = FLCore::refpts(x),
  statistics = .validStatistics(om(x)),
  metrics = NULL,
  years = dimnames(om(x))$year[-1],
  om = name(x@om),
  type = "MP",
  run = "1",
  control = FALSE,
  ...
)
```

```
## S4 method for signature 'FLmses'
performance(x, type = NULL, refpts = NULL, ...)

## S4 method for signature 'list'
performance(x, statistics, refpts = NULL, years = "missing", ...)

## S4 method for signature 'FLStock'
performance(
  x,
  statistics,
 metrics = list(R = rec, SB = ssb, B = tsb, C = catch, L = landings, D = discards, F =
    fbar, HR = hr),
  ...
)

## S4 method for signature 'FLStocks'
performance(x, statistics, ...)
```

Arguments

x	An object holding simulation results. Supported classes: FLQuants, FLStock, FLStocks, FLom, FLmse, FLmses, or list.
statistics	A list of statistics to compute. Each element must be a named list with a formula and metadata (name, desc). See Details.
refpts	Reference points for calculations, typically an FLPar object. Defaults to the reference points held by inputs that contain them. For FLmses and lists, a supplied FLPar applies to every member, while a named list supplies one object per member. FLQuants defaults to an empty FLPar().
years	Years on which statistics should be computed. Can be: • A vector of years to use • A named list of year vectors (names become year labels in output) Defaults to last year of input if omitted.
om	Optional name for the operating model.
type	Optional name for the MP type.
run	Optional name for the model run.
mp	Optional combined MP name. Auto-generated if not provided.
...	Additional arguments passed through (e.g., custom metrics, tracking data).
metrics	Optional metrics object for FLStock/FLStocks input. Can be: • An FLQuants object with pre-computed metrics • A list of metric functions • A single function to compute metrics
control	Logical. For FLmse input, include HCR control arguments in output? Defaults to FALSE.

Details

Each statistic is defined as a named list containing:

- A formula (unnamed element) using metric and reference point names, e.g., `~yearMeans(SB/SB0)`
- name: Short name for tables/plots, e.g., "SB/SB0"
- desc: Longer description, e.g., "Mean spawner biomass relative to unfished"

Statistics formulas can reference:

1. Names of `FLQuants` elements (metrics from estimation)
2. Parameter names in the `refpts` object
3. FLQuant dimension names (age, year, unit, season, area)
4. Functions callable on the source object (for non-FLQuants input)

Value

A `data.table` containing computed performance statistics with columns:

- statistic: Name of the computed statistic
- year: Year or period for which statistic was computed
- name: Display name of statistic
- desc: Description of statistic
- iter: Iteration number (or 1 if statistic computes a probability)
- data: The computed value
- om, type, run, mp: Identifiers for the analysis

Author(s)

Iago Mosqueira (WMR)

See Also

statistics, `refpts()`, `metrics()`

Examples

```
# LOAD example FLmse object
data(plesim)
# Extract FLQuants using metrics
x <- metrics(om)
performance(x, statistics=statistics[c("SB", "SBMSY", "F", "FMSY")],
  refpts=refpts(om), om="ple", run="r00", type="test")
# Compute on OM, name taken from slot
performance(om, statistics=statistics[c("SB", "SBMSY", "F", "FMSY")],
  run="r00", type="test")
# Setup an example MSE
control <- mpCtrl(list(
  est = mseCtrl(method=perfect.sa),
```

```
  hcr = mseCtrl(method=fixedF.hcr, args=list(ftrg=0.15))))
# ... and run it
mse <- mp(om, ctrl=control, args=list(iy=2025, fy=2030))
# Compute performance using all default statistics, data(statistics)
performance(mse, run="r00", type="test")
# or select a few of them
performance(mse, statistics=statistics[c("SBMSY", "FMSY")], run="r00", type="test")
```

`plesim` *FLom object*

Description

Aliquam sagittis feugiat felis eget consequat. Praesent eleifend dolor massa, vitae faucibus justo lacinia a. Cras sed erat et magna pharetra bibendum quis in mi. Sed sodales mollis arcu, sit amet venenatis lorem fringilla vel. Vivamus vitae ipsum sem. Donec malesuada purus at libero bibendum accumsan. Donec ipsum sapien, feugiat blandit arcu in, dapibus dictum felis.

Format

An object of class *FLom*.

`plot_hockeystick.hcr` *Plot a hockey-stick harvest control rule*

Description

Plots the shape of a hockey-stick HCR, as implemented by `hockeystick.hcr`, from a set of arguments or an `mseCtrl`/`mpCtrl` object.

Usage

```
plot_hockeystick.hcr(
  args,
  obs = NULL,
  kobe = FALSE,
  xtarget = args$trigger,
  alpha = 0.3,
  xlim = max(trigger) * 1.5,
 labels = c(lim = "limit", trigger = "trigger", min = "min", target = "target", drop =
    "drop")
)
```

Arguments

args	A list of HCR arguments (as used by hockeystick.hcr), or an mseCtrl or mpCtrl object from which those arguments are extracted.
obs	An optional FLStock from which historical values of metric and output are computed and added to the plot.
kobe	If TRUE, a Kobe-style colour background (red/yellow/green) is added to the plot, logical.
xtarget	The actual target value on the metric axis (e.g. Btarget), used to draw the Kobe background, numeric. Defaults to args$trigger.
alpha	Transparency level applied to the Kobe background polygons, numeric.
xlim	Upper limit of the x-axis (metric), numeric. Defaults to 1.50 times trigger.
labels	A named character vector of labels for lim, trigger, min, target and drop, used to annotate the plot. Set to NULL to omit labels.

Details

The plot shows the control variable (output) against the stock metric, with dashed lines and labels marking the lim, trigger, min, target and, when set, drop reference points. If obs is given, an FLStock, the historical trajectory of metric and output is added to the plot. If kobe = TRUE, a Kobe-style background of red, yellow and green zones is added, computed with respect to xtarget, the actual target value on the metric axis (e.g. Btarget), which defaults to the trigger value.

Value

A ggplot object.

Examples

```
data(ple4)

# Set example HCR arguments
args <- list(lim=1e5, trigger=4e5, target=0.25, min=0,
  metric="ssb", output="fbar")

# Plot hockeystick.hcr for given arguments
plot_hockeystick.hcr(args)

# Plot wheh 'drop' has been set
args <- list(lim=0, trigger=4e5, target=0.25, min=0, drop=2e5,
  metric="ssb", output="fbar")

plot_hockeystick.hcr(args)

# Add metric and output from FLStock
plot_hockeystick.hcr(args, obs=ple4)

# Superpose Kobe colours
plot_hockeystick.hcr(args, kobe=TRUE)
```

```
data(ple4)

# Change labels
plot_hockeystick.hcr(args, obs=ple4, kobe=TRUE,
  labels=c(limit="Blim", trigger="Btrigger", target="Ftarget"))

# Set actual x (e.g. biomass) target.
plot_hockeystick.hcr(args, obs=ple4, kobe=TRUE, xtarget=args$trigger * 0.80)

#' Add line and label for Btarget
plot_hockeystick.hcr(args, obs=ple4, kobe=TRUE, xtarget=args$trigger * 0.80) +
geom_vline(xintercept=args$trigger * 0.80, linetype=3) +
geom_label(x=args$trigger * 0.80, y=0.7, label="SBtarget")

# ADD decade labels
plot_hockeystick.hcr(args, obs=ple4, kobe=TRUE) +
  geom_label(data=model.frame(metrics(ple4[, ac(seq(1957,2017, by=10))],
  list(metric=ssb, output=fbar))), aes(label=year),
  fill=c("white", rep("gray", 5), "orange"))

# Example on relative terms where trigger < xtarget
args <- list(lim=0., trigger=0.9, target=1, min=0,
  metric="ssb", output="fbar")

plot_hockeystick.hcr(args, kobe=TRUE, xtarget=1) +

geom_vline(xintercept=1)
```

---

sampling.oem *sampling.oem*

Description

Samples from an operating model to obtain catch, biology and abundance data

Usage

```
sampling.oem(
  stk,
  deviances,
  observations,
  stability = 1,
  wts = TRUE,
  eps = .Machine$double.eps,
  args,
  tracking
)
```

Arguments

stk	An FLStock object as obtained by the call to *stock(om)*.
deviances	A named list of deviances, see Details.
observations	A named list of observations, see Details.
stability	Stability of the index survey, passed to *survey()*.
wts	Should the weights-at-age in the stock be updated?
eps	Small constant added to stock deviances.
args	Options and arguments passed on by *mp()*.
tracking	The tracking object.

Details

This observation error model (OEM) function mimics the most common data collection regime, in which catch-at-age and biology is sampled from the population, and one or more indices of abundance are derived from surveys or CPUE data.

The FLStock object passed to *sampling.oem* by the *mp* function is simplified to match the dimensions of that present in the *observations* slot.

Value

A named list with elements *stk*, *idx*, *observations* and *tracking*.

Author(s)

Iago Mosqueira (WUR) & Ernesto Jardim (MSC).

See Also

mp

Examples

```
data(plesim)
# Generate samples from year 2000:2016
sampling.oem(stock(om), deviances=deviances(oem),
  observations=observations(oem),
  args=list(y0=2000, iy=2020, dy=2021, dys=2021, frq=1), tracking=FLQuant())
```

`selectMetric` *Select and/or Compute a Metric from the* stk *and* ind *inputs*

Description

A metric, defined here as a time series, commonly age-aggregated, is computed or extracted from the input FLStock (*stk*) and FLQuants (*ind*). These have been returned by the call to the *est*inmation step in a call to `mp()`.

Usage

```
selectMetric(metric = "missing", stk, ind, ...)
```

Arguments

`metric`	A metric to use, which can be one of the following: • `missing`: Defaults to the first element of the `ind` object if only one element is present. • `character`: The name of a metric in `ind` to extract, or the name of a function to compute the metric. • `function`: A function to compute the metric with *stk* as input.
`stk`	The stock object, used for computing the metric (if applicable).
`ind`	A FLQuants object containing potential metrics, used for extraction based on `metric`.
`...`	Additional arguments passed to the function `metric` (if it is a function or callable).

Details

If *metric* is a character string and matches a name in *ind*, then that *FLQuant* element is returned. Otherwise, or if *metric* is a function, it is called on *stk*. See examples below.

Value

The selected or computed metric, either extracted from `ind` or computed using `stk` and `metric`.

Author(s)

Iago Mosqueira (WUR)

Examples

```
data(ple4)
# Computes 'catch' metric from 'stk', 'ind' is empty
selectMetric("catch", stk=ple4, ind=FLQuants())

# Computes own rfunction (ratio of discards to landings) as metric from 'stk'
selectMetric(function(x) discards(x) / landings(x), stk=ple4, ind=FLQuants())
```

```
# Returns 'catch' metric from 'ind' (defined as log), takes precedence over 'stk'
selectMetric("catch", stk=ple4, ind=FLQuants(catch=log(catch(ple4))))

# Any function available for 'stk' works
selectMetric(ssb, stk=ple4, ind=FLQuants())
```

`setcontrol` *setcontrol Modify on the fly the arguments on any mpCtrl module*

Description

DEFINITION

Usage

```
setcontrol(control, ...)
```

Examples

```
# Example dataset
data(plesim)

# Sets up an mpCtrl using hockeystick(fbar~ssb)
ctrl <- mpCtrl(est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(metric="ssb", trigger=14000,
    output="fbar", target=0.18)))

# Runs mp between 2021 and 2035
run <- mp(om, control=ctrl, args=list(iy=2021, fy=2035))

# Alters the value for the 'trigger' HCR argument
run02 <- mp(om, control=setcontrol(ctrl, hcr=list(trigger=15000)),
  args=list(iy=2021, fy=2035))

# Plots results
plot(om, list(T14k=run, T15k=run02))
```

`split.is` *Split a biol target across fleets by setting relative 'quant' proportions*

Description

Split a biol target across fleets by setting relative 'quant' proportions

Usage

```
split.is(stk, ctrl, split, quant = unique(ctrl$quant)[1], args, tracking)
```

Examples

```
data(ple4)
```

`statistics` *A complete set of performance statistics*

Description

A list containing a large number of performance statistics:

A list of key statistics used in fisheries management to evaluate the performance of management procedures. This is a long list that contains statistics that measure very similar outcomes, so a subset should be made of those most relevant to the management objectives of interest.

Usage

```
data(statistics)

data(statistics)
```

Format

A named list with elements of class list, each containing three elements: a formula, a long name ('name'), and a description ('desc'), both of type 'character':

SB SB: Mean spawner biomass

SB0 SB/SB[0]: Mean spawner biomass relative to unfished

minSB0 min(SB/SB[0]): Minimum spawner biomass relative to unfished

SBMSY SB/SB[MSY]: Mean spawnwer biomass relative to SBMSY

F F: Mean fishing mortality

Ftarget F/F[target]: Mean fishing mortality relative to target

FMSY F/F[MSY]: Mean fishing mortality relative to FMSY

green P(Green): Probability of being in Kobe green quadrant

orange P(Orange): Probability of being in Kobe orange quadrant

yellow P(Yellow): Probability of being in Kobe yellow quadrant

red P(Red): Probability of being in Kobe red quadrant

PSBMSY P(SB>=SB[MSY]): Probability of SB greater or equal to SBMSY

PSBlim P(SB>SB[limit]): Probability that spawner biomass is above SBlim

PSB20B0 P(SB > 0.20 %*% SB[0]): Probability that spawner biomass is above 20% SB0

risk1 mean(P(SB<B[limit])): ICES Risk 1, mean probability that spawner biomass is below Blim

risk2 once(P(SB<B[limit])): ICES Risk 2, probability that spawner biomass is above Blim once

risk3 max(P(SB>B[limit])): ICES Risk 3, max probability that spawner biomass is above Blim

C mean(C): Mean catch over years

CMSY C/MSY: Mean proportion of MSY

AAVC AAV(C): Average annual variability in catch

IACC IAC(C): Percentage inter-annual change in catch

PC0 P(shutdown): Probability of fishery shutdown, defined as catch less than 10% of MSY

A list containing named elements, each of which represents a specific statistic. Each of them contains:

- `formula`: A formula defining how the metric is calculated.
- `name`: A short descriptive name. This can contain `plotmath()` expressions to be parsed by plot functions.
- `desc`: A more detailed description of the metric.

Details

Performance statistics are used by the `performance()` method to compute time series, or aggregates along time, of quantities of interest related to the result of applying a particular management procedure to an operating model. They combione `metrics()` computed from the projected stocks, populations and fisheries, with biological, economic or other reference points, but can also use the results of calculations and decisions carried out by the MP.

Of the three elements in the list used to define each statistic, the first unnamed element, of class 'formula' is the one evaluated by `performance()`. The formula is evaluated with access to the reference points of the OM, contained in the `refpts` slot, a set of metrics obtained from the projected OM, the contents of the tracking table with decisions and outputs from the MP internal calcultions, as well as any function available in the workspace.

The statistics currently included are:

SB Spawner biomass in tonnes (SB), from the `SB` metric.

SB0 Spawner biomass relative to unfished ($SB/SB0$), requires the `SB0` refpt.

minSB0 Minimum spawner biomass relative to unfished ($min(SB/SB0)$) requires the `SB0` refpt..

SBMSY Spawner biomass relative to $SB[MSY]$ ($SB/SB[MSY]$) requires the `SBMSY` refpt..

R Recruitment (R), from the `R` metric.

F Fishing mortality (F), from the `F` metric.

Ftarget Fishing mortality relative to target ($F/F[target]$) requires the `Ftarget` refpt.

FMSY Fishing mortality relative to $F[MSY]$ ($F/F[MSY]$) requires the `FMSY` refpt.

green Probability of being in the Kobe green quadrant ($P(Green)$), requires the `SBMSY` and `FMSY` refpts.

orange Probability of being in the Kobe orange quadrant ($P(Orange)$), requires the `SBMSY` and `FMSY` refpts.

yellow Probability of bein, requires the `SBMSY` and `FMSY` refptsg in the Kobe yellow quadrant ($P(Yellow)$), requires the `SBMSY` and `FMSY` refpts. #'

red Probability of being in the Kobe red quadrant ($P(Red)$), requires the `SBMSY` and `FMSY` refpts.

PSBMSY Probability that spawner biomass is greater than or equal to $SB[MSY]$ ($P(SB >= SB[MSY])$), requires the SBMSY refpt.

PSBlim Probability that spawner biomass is above $SB[lim]$ ($P(SB > SB[lim])$), requires the SBlim refpt.

PSB20B0 Probability that spawner biomass is above 20% of $SB0$ ($P(SB > 0.20\% * \%SB0)$), requires the SB0refpt.

risk1 ICES Risk 1: Probability that spawner biomass is below $B[lim]$ ($P(SB < B[lim])$), requires the SBlim refpt.

risk2 ICES Risk 2: Probability that spawner biomass falls below $B[lim]$ at least once ($once(P(SB < B[lim]))$), requires the SBlim refpt.

risk3 ICES Risk 3: Maximum probability that spawner biomass is below $B[lim]$ ($max(P(SB < B[lim]))$), requires the SBlim refpt.

C Catch in tonnes ($C[t]$), from the C metric.

CMSY Proportion of maximum sustainable yield (C/MSY), requires the MSY refpt.

IACC Percentage inter-annual change in catch ($IAC(C)$), from the C metric.

PIACC20 Probability that the inter-annual change in catch being less than 20% ($P(IAC(C) < 0.20)$), from the C metric.

PC0 Probability of fishery shutdown ($P(shutdown)$), defined as catch falling below 10% of MSY, so requires the MSY refpt.

Examples

```
data(statistics)
# Access a specific statistic
statistics$SBMSY
```

`tac.is` *TAC implementation system module*

Description

Performs a short term forecast (STF) for the target fishing mortality to obtain the corresponding catch.

Usage

```
tac.is(
  stk,
  ctrl,
  args,
  output = "catch",
  recyrs = -2,
  fsqyrs = 1,
  Fdevs = unitMeans(fbar(fut)) %=% 1,
```

```
  dlow = NA,
  dupp = NA,
  fmin = 0,
  reuse = TRUE,
  initac = unitSums(metrics(stk, output)[, ac(iy - data_lag)]),
  tracking
)
```

Arguments

stk	The perceived FLStock.
ctrl	The fwdControl output by the *hcr* step, target must be 'fbar'.
args	The MSE run arguments.
recyrs	Years to use for geometric mean recruitment if projection. Defaults to all years minus the last two.
Fdevs	Deviances on the fbar input to incorporate error and bias when MP is run using the pseudo-estimators 'perfect.sa' or 'shortcut.sa'.
dlow	Limit to decreases in output catch, as a proportional change (0.85 for 15%). Applied only when metric > lim, as set by 'hcr' step.
dupp	Limit to increases in output catch, as a proportional change (1.15 for 15%). Applied only when metric > lim, as set by 'hcr' step.
fmin	Minimum fbar to apply when catch change limits are use.
initac	Initial catch from which to compute catch change limits. Defaults to previous observed catch.
tracking	The tracking object.

Details

A `fwdControl` object obtained from the 'hcr' step is applied in the management year (`ay + mlag`) or years (`seq(ay + mlag, ay + mlag + freq)`. An assumption is made on the mortality in the assessment year (`ay`), which becomes the intermediate year in this projection. By default this is set to Fbar = Fsq, that is, the same fishing mortality estimated in the last data year (`ay - data_lag`).

The projection applies a constant recruitment, equal to the geometric mean over an specified number of years. By default all years minus the last two are included in the calculation. An specific set of years can be employed, by specifying a character vector of year names, or two values can be given for the number of years to be inlcuded, counting from the last, and how many years to exclude at the end. For example, `c(30, 2)` will use the last 30 years but excluding the last two, usually worst estimated.

Examples

```
data(plesim)
# Setup control with tac.is
control <- mpCtrl(list(est=mseCtrl(method=perfect.sa),
  hcr=mseCtrl(method=hockeystick.hcr,
    args=list(lim=0, trigger=14000, target=0.18)),
```

```
  isys=mseCtrl(method=tac.is, args=list(recyrs=-3, fnsqy=3, output='landings'))))
# Run MP until 2025
run <- mp(om, oem, ctrl=control, args=list(iy=2021, fy=2027))
# Plot run time series
plot(om, TAC.IS=run)
```

`tunebisect` *Tune Management Procedures Using Bisection Method*

Description

The `tunebisect` function is designed to tune a single parameter of a single modules in a Management Procedure (MP), typically in the Harvest Control Rule (hcr). It uses a bisection algorithm to iteratively adjust the parameter value unitl it achieves specified probability value for a given management objective overn a selected time frame. For example, it can try to find a value for the `target` argument of a `hocketstick.hcr` that obtains a 60% mean probability of SSB being above SSBMSY over the 2032 to 2042 period.

Usage

```
tunebisect(
  om,
  oem = NULL,
  iem = NULL,
  control,
  statistic,
  metrics = NULL,
  args,
  tune,
  prob = 0.5,
  tol = 0.01,
  maxit = 12,
  years = ac(seq(args$iy + 1, args$fy - 1)),
  parallel = TRUE,
  verbose = TRUE,
  window = TRUE,
  ...
)
```

Arguments

`om`	An object representing the Operating Model, of class FLom or FLombf.
`oem`	Observation Error Model, class FLoem or missing, the default.
`control`	A control object containing the settings and parameters for the Management Procedure, class mpCtrl.
`statistic`	A named list of length 1 defining the statistic used for tuning. The list must include:

name The name of the statistic.

formula The formula used to compute the statistic.

desc A description of the statistic.

`metrics` Optional. A set of metrics used for performance evaluation; defaults to `NULL` to use the default metrics for the class of the `om` object.

`args` A list of arguments for running the Management Procedure. Must include `iy` (initial year).

`tune` A named list specifying the parameter of the HCR to tune and its range. The list must include the parameter name and the minimum and maximum values as a vector of length 2.

`prob` The target probability for tuning; defaults to `0.5`. Must be a value between 0 and 1.

`tol` Tolerance for the difference between the computed and target probabilities; defaults to `0.01`.

`maxit` Maximum number of iterations, defaults to 12.

Details

Two initial MP runs are carried out employing the argument values provided in `tune`. They need to return probabilities that are on both sides of the expected one. If not, the function will return both runs for inspection. The function should then be rerun with an wider range of values, or the time-frame extended.

References

Burden, Richard L.; Faires, J. Douglas (2016), "2.1 The Bisection Algorithm", Numerical Analysis (10th ed.), Cenage Learning, ISBN 978-1-305-25366-7

Examples

```
# dataset contains both OM (FLom) and OEM (FLoem)
data(plesim)
# choose sa and hcr
control <- mpCtrl(list(
  est = mseCtrl(method=perfect.sa),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(lim=0,
  trigger=14000, target=0.18))))
# load statistics
data(statistics)
tun <- tunebisect(om, oem=oem, control=control, args=list(iy=2021, fy=2035),
tune=list(target=c(0.12, 0.32)),
metrics=list(SB=ssb), statistic=statistics['PSBMSY'], years=2025:2034)
# Plot tuned MP
plot(om, tun)
```

Index

∗ **classes**
FLom, 14
∗ **datasets**
plesim, 38
statistics, 44
∗ **function**
sampling.oem, 40
∗ **manip**
indicator.is, 21
∗ **methods**
cpues.ind, 4
debug-mse, 6
∗ **utilities**
performance, 34

0, *37*, *44*

append,FLombf,FLombf-method *(*FLombf*)*, 17
args *(*mseCtrl-class*)*, 30
args,FLmse-method *(*FLmse-class*)*, 11
args,mpCtrl-method *(*mpCtrl-class*)*, 24
args,mseCtrl-method *(*mseCtrl-class*)*, 30
args-methods *(*mseCtrl-class*)*, 30
args<- *(*mseCtrl-class*)*, 30
args<-,FLmse,list-method *(*FLmse-class*)*, 11
args<-,mpCtrl,ANY-method *(*mpCtrl-class*)*, 24
args<-,mpCtrl-method *(*mpCtrl-class*)*, 24
args<-,mseCtrl,list-method *(*mseCtrl-class*)*, 30
args<--methods *(*mseCtrl-class*)*, 30

combine,FLoem,FLoem-method *(*FLoem-class*)*, 13
combine,FLom,FLom-method *(*FLom*)*, 14
control,FLmse-method *(*FLmse-class*)*, 11
control<-,FLmse,mpCtrl-method *(*FLom*)*, 14
cpue.ind, 2
cpues.ind, 4

debug, *7*
debug,FLo,ANY-method *(*debug-mse*)*, 6
debug,mpCtrl,character-method *(*mpCtrl-class*)*, 24
debug,mseCtrl,missing-method *(*debug-mse*)*, 6
debug-mse, 6
deviances,FLoem-method *(*FLoem-class*)*, 13
deviances<-,FLoem,list-method *(*FLoem-class*)*, 13

est,mpCtrl-method *(*mpCtrl-class*)*, 24
est<- *(*mpCtrl-class*)*, 24
est<-,mpCtrl,function-method *(*mpCtrl-class*)*, 24
est<-,mpCtrl,list-method *(*mpCtrl-class*)*, 24
est<-,mpCtrl,mseCtrl-method *(*mpCtrl-class*)*, 24
est<--methods *(*mpCtrl-class*)*, 24
exists *(*mseCtrl-class*)*, 30
exists,mseCtrl-method *(*mseCtrl-class*)*, 30

fixedC.hcr, 7
fixedF.hcr, 8
FLComp, *16*
FLiem *(*FLiem-class*)*, 10
FLiem-class, 10
FLiem-methods *(*FLiem-class*)*, 10
FLmse *(*FLmse-class*)*, 11
FLmse-class, 11
FLmse-methods *(*FLmse-class*)*, 11
FLmses, *29*
FLoem, *48*
FLoem *(*FLoem-class*)*, 13
FLoem-class, 13
FLoem-methods *(*FLoem-class*)*, 13
FLom, 14, *48*
FLom-class *(*FLom*)*, 14

FLom-methods *(FLom)*, 14
FLombf, 17, *48*
FLombf-methods *(FLombf)*, 17
fwd.om, 18

hcr,mpCtrl-method *(mpCtrl-class)*, 24
hcr<- *(mpCtrl-class)*, 24
hcr<-,mpCtrl,function-method *(mpCtrl-class)*, 24
hcr<-,mpCtrl,list-method *(mpCtrl-class)*, 24
hcr<-,mpCtrl,mseCtrl-method *(mpCtrl-class)*, 24
hcr<--methods *(mpCtrl-class)*, 24
hockeystick.hcr, 18
hr,FLombf-method, 20

index.hat,FLIndexBiomass,FLStock-method, 21
indicator.is, 21
initialize,FLiem-method *(FLiem-class)*, 10
initialize,mpCtrl-method *(mpCtrl-class)*, 24
initialize,mseCtrl-method *(mseCtrl-class)*, 30
isys,mpCtrl-method *(mpCtrl-class)*, 24
isys<- *(mpCtrl-class)*, 24
isys<-,mpCtrl,function-method *(mpCtrl-class)*, 24
isys<-,mpCtrl,list-method *(mpCtrl-class)*, 24
isys<-,mpCtrl,mseCtrl-method *(mpCtrl-class)*, 24
isys<--methods *(mpCtrl-class)*, 24
iter,FLiem-method *(FLiem-class)*, 10
iter,FLoem-method *(FLoem-class)*, 13
iter,mpCtrl-method *(mpCtrl-class)*, 24
iters,mpCtrl-method *(mpCtrl-class)*, 24

method *(mseCtrl-class)*, 30
method,mpCtrl-method *(mpCtrl-class)*, 24
method,mseCtrl-method *(mseCtrl-class)*, 30
method-methods *(mseCtrl-class)*, 30
method<- *(mseCtrl-class)*, 30
method<-,mpCtrl,function-method *(mpCtrl-class)*, 24
method<-,mpCtrl-method *(mpCtrl-class)*, 24
method<-,mseCtrl,function-method *(mseCtrl-class)*, 30
method<--methods *(mseCtrl-class)*, 30
metrics, *45*
metrics(), *37*, *45*
mp, *5*, 22, *29*, *41*
mpCtrl, *29*, *48*
mpCtrl *(mpCtrl-class)*, 24
mpCtrl-class, 24
mpCtrl-methods *(mpCtrl-class)*, 24
mpDispatch, 27
mps, 28
mseCtrl *(mseCtrl-class)*, 30
mseCtrl-class, 30
mseCtrl-methods *(mseCtrl-class)*, 30
MSY, *48*

nbrOfWorkers, *29*

observations *(FLoem-class)*, 13
observations,FLoem-method *(FLoem-class)*, 13
observations-methods *(FLoem-class)*, 13
observations<- *(FLoem-class)*, 13
observations<-,FLoem,ANY,FLStock-method *(FLoem-class)*, 13
observations<-,FLoem,missing,list-method *(FLoem-class)*, 13
observations<--methods *(FLoem-class)*, 13
oem *(FLmse-class)*, 11
oem,FLmse-method *(FLmse-class)*, 11
oem-methods *(FLmse-class)*, 11
oem<- *(FLmse-class)*, 11
oem<-,FLmse,FLoem-method *(FLom)*, 14
oem<--methods *(FLmse-class)*, 11
om *(FLmse-class)*, 11
om,FLmse-method *(FLmse-class)*, 11
om-methods *(FLmse-class)*, 11
om<- *(FLmse-class)*, 11
om<-,FLmse,FLo-method *(FLom)*, 14
om<--methods *(FLmse-class)*, 11

partialHR, 32
perfect.oem, 33
perfect.sa, 34
performance, *29*, 34
performance(), *45*

performance,FLmse-method *(performance)*, 34
performance,FLmses-method *(performance)*, 34
performance,FLo-method *(performance)*, 34
performance,FLombf-method *(performance)*, 34
performance,FLQuants-method *(performance)*, 34
performance,FLStock-method *(performance)*, 34
performance,FLStocks-method *(performance)*, 34
performance,list-method *(performance)*, 34
phcr,mpCtrl-method *(mpCtrl-class)*, 24
phcr<- *(mpCtrl-class)*, 24
phcr<-,mpCtrl,function-method *(mpCtrl-class)*, 24
phcr<-,mpCtrl,list-method *(mpCtrl-class)*, 24
phcr<-,mpCtrl,mseCtrl-method *(mpCtrl-class)*, 24
phcr<--methods *(mpCtrl-class)*, 24
plesim, 38
plot_hockeystick.hcr, 38
plotmath(), *45*

refpts(), *37*

sampling.oem, 40
seasonal.is, *22*
selectMetric, 42
setcontrol, 43
show,FLoem-method *(FLoem-class)*, 13
show,mpCtrl-method *(mpCtrl-class)*, 24
show,mseCtrl-method *(mseCtrl-class)*, 30
smooth_index, *5*
sp.is, *22*
split.is, 43
sr,FLom-method *(FLom)*, 14
sr<-,FLom,FLSR-method *(FLom)*, 14
statistics, *37*, 44
stock,FLom-method *(FLom)*, 14
stock<-,FLom,FLStock-method *(FLom)*, 14
summary,mseCtrl-method *(mseCtrl-class)*, 30

tac.is, *22*, 46
tm,mpCtrl-method *(mpCtrl-class)*, 24
tm<- *(mpCtrl-class)*, 24
tm<-,mpCtrl,function-method *(mpCtrl-class)*, 24
tm<-,mpCtrl,list-method *(mpCtrl-class)*, 24
tm<-,mpCtrl,mseCtrl-method *(mpCtrl-class)*, 24
tm<--methods *(mpCtrl-class)*, 24
tracking, *45*
tracking *(FLmse-class)*, 11
tracking,FLmse-method *(FLmse-class)*, 11
tracking-methods *(FLmse-class)*, 11
tracking<- *(FLmse-class)*, 11
tracking<-,FLmse,data.table-method *(FLom)*, 14
tracking<--methods *(FLmse-class)*, 11
tunebisect, 48

undebug,FLo,ANY-method *(debug-mse)*, 6
undebug,mpCtrl,character-method *(debug-mse)*, 6
undebug,mpCtrl,missing-method *(debug-mse)*, 6
undebug,mseCtrl,missing-method *(debug-mse)*, 6

weighted.mean, *5*

zscore, *5*